Transbase®
System Guide

Transaction Software GmbH
Willy-Brandt-Allee 2
D-81829 München
Germany
Phone: +49-89-62709-0
Fax: +49-89-62709-11
Email: info@transaction.de
http://www.transaction.de

Version 6.8.1.40
November 02, 2010

Contents

Chapter 1

Overview of Transbase Manuals

The following is a list and a brief description of all Transbase manuals.

Getting Started This manual is a quick introduction into some basic functionality and concepts of Transbase.

System Guide This manual describes the commands and tools for the Transbase Database Administrator and many internal functions and concepts .

SQL Reference Manual Contains the complete TB/SQL kernel language definition.

Transbase Embedded SQL Describes the standardized Embedded SQL programming interface (EXEC SQL ...) in a C language environment.

Programming Interface TBX Describes the native C programming interface to Transbase provided by the library tbx.a and tbx.dll, resp.

TransbaseCD Guide Comprises the description of the unique Transbase CD-ROM Databases. Described are the preparation, processing and tuning (e.g. loading and compression) of CD-ROM databases.

Command Control Language CCL An easy-to-learn yet powerful programming languange which offers a comfortable SQL interface to the Transbase kernel.

Transbase Interactive Interface TBI Describes the line oriented interactive interface TBI and its table editor reledit.

Transbase Query Optimization and Locking Guide Describes to a far extent how the Transbase query optimizer works and how it can be influenced by commands and additional language features.

UFI User's Guide (only Unix Platforms) Describes the screen oriented interactive interface UFI.

Chapter 2

Transbase-Packages

The product Transbase/CD consists of several software packages. The following are offered on Windows platforms:

- Transbase / Transbase-CD - Server
- Myriad / Myriad-CD - Server
- Application Development Toolkit (TBX / ESQL)
- CD-ROM Developers Toolkit
- Frontend Tools (Ufi, tbi, ccl)

Each of these packages is installed with a separate license number which can be acquired from TransAction Software and is requested during the installation.

For Windows platforms three other packages are available:

- Transbase ODBC Driver (included in ISQL or Developers
- Transbase .NET Driver (included in ISQL or Developers
- Transbase OLEDB provider (included in ISQL or Developers Toolkit)

Chapter 3

Overview of Transbase Platforms

3.1 Transbase on Unix Platforms

Transbase is available for various Unix platforms. The updated list can be ordered at Transaction Software GmbH in Munich, Germany.

3.2 Transbase for Windows

This chapter presents an overview about the supported Windows platforms, the Transbase software packages and the implemented communication protocols. It also introduces notations used in the more detailed representations of the successive chapters.

3.2.1 Overview

3.2.2 Transbase for WindowsNT

The 32 bit variant of Transbase runs on WindowsNT. The available communication protocols are listed in the table.

3.2.3 Transbase for Win32S

The 32 bit variant of Transbase runs on Win32S based on Windows 3.1 (WIN32) or Windows for Workgroups 3.11(WfW32). The available communication protocols are listed table.

Protocol	Only WfW	Needed Software
TCP/IP	Yes	TCP/IP for WfW
TCP/IP	yes	TCP32 for WfW
TCP/IP	no	PC/NFS V5.0 or higher
TCP/IP	no	LanWorkPlace V4 or higher
TCP/IP	no	PCTCP 3.0
TCP/IP	no	any winsock compatible
local DDE	no	-
Network DDE	yes	-
Decnet	no	Pathworks
Linked_In	no	-

Table 3.1: Communication Protocols on Windows 3.1/3.11

Protocol	Needed Software
TCP/IP	-
DDE	-
Network DDE	-
Linked_In	-

Table 3.2: Communication Protocols on Windows/NT

Note: Network DDE is *not* availabe for WfW32. TCP/IP is available if the installed TCP/IP stack supports a winsock Interface.

Appendix B shows the relationship between Windows platforms and available communications protocols.

Protocol	Only WfW32	Needed Software
TCP/IP	yes	TCP/IP for WfW
TCP/IP	yes	TCP32 for WfW
TCP/IP	no	PC/NFS V5.0 or higher
TCP/IP	no	LanWorkPlace V5 or higher
TCP/IP	no	PCTCP 3.0
TCP/IP	no	any winsock compatible
local DDE	no	-
Linked_In	no	-

Table 3.3: Communication Protocols on Windows/32S

3.3 Transbase for Novell Netware

Following packages are available for Novell Netware V3.12 and for Novell Netware V4.1

- Transbase / Transbase-CD - Server
- Myriad / Myriad-CD - Server
- CD-ROM Developers Toolkit

TCP/IP must be installed on the Netware server to make the server accessible from the client machines.

Chapter 4

General Concepts of Transbase

Transbase is a multiple distributed database system. The term "multiple" means that applications can access more than one database within a session. Those databases may reside not only on the local host but also on remote hosts. Furthermore, distributed queries are supported. An application can access several databases via one query within one single connection. Details are explained in a separate section below.

Transbase thus is called a distributed system because it provides distributed transactions and distributed queries. Data consistency and transaction consistency is provided, even in the case of distributed access and multiple updates. Note that location transparency is not provided. Database names must be provided for multiple CONNECTs or, for distributed queries, in location suffixes for tablenames.

Transbase provides for data security by a clean implementation of transactions both in local or distributed operation.

Full multi-user capability is provided by splitting Transbase into independent processes - one for each application program - and a shared memory region which serves as communication area between those processes.

To guarantee data consistency, Transbase employs locks on database objects which are to be shared between concurrent transactions. Deadlocks are detected and broken by Transbase, both in local and distributed operation. Indefinite delay is prevented by a timeout mechanism which is of course user-definable.

Data privacy is established by granting privileges on database objects to users. A database administrator is introduced as the owner of a database. He grants access privileges to users. Users have to login into a database by supplying their user name and a password which is checked by Transbase.

The following chapters describe the concepts of a database, the user and privileges concept and the transaction concept in detail.

4.1 Databases

Transbase databases are autonomous. By autonomy we understand that each database can be run independently of others. Especially, each database has its own set of tables, its own name space, and its own user community, and, of course, its own files and directories.

Transbase databases are concurrently accessible by local or remote clients which may be located on heterogeneous machines or even systems. Transbase uses TCP/IP communication software for remote and local access.

Databases are named by a logical name which is unique on the local host. When a database is accessed from a remote client, it is identified by the hostname of the database server and the unique local name.

Databases are accessed by statements (e.g. a `SELECT` statement).

An application has to `CONNECT` to databases it wants to access; a connection is simply a communicatiion link to a database server. After connection, the application program (or the user, resp.) has to be authorized before the first statement is accepted.

4.2 Users and Privileges

To establish data privacy in a Transbase database, each user (or application program resp.) has to login into the database and have his authorization checked before being able to access any table. The authorization check is based on a user-changeable password. Passwords are stored within the database in encrypted form only (see the chapter "The sysuser Table" below).

Upon creation of a database the database administrator (DBA) is automatically installed as user with the user name "tbadmin" and an empty password. It is strongly recommended to change the DBA password immediately after creation. (See *AlterPasswordStatement* in TB/SQL-Manual)

Other database users can be added or deleted by the DBA only. The database administrator is responsible for choosing a unique user name (which may be different from the operating system user identification) and a user class for a new user.

To protect machine resources and database objects against misuse, Transbase applies three concepts with the notions userclass, owner and access privilege.

The concept of userclass applies to users. Each user has a userclass which can be changed by the DBA only. (See Grant-Userclass-Statement, Revoke-Userclass -Statement). The userclass regulates the right to login, to create objects (tables,

views, indexes) and also defines whether a user has the property of a DBA. The following four userclasses are supported:

Userclass	**Access Right**
"no access"	No login
"access"	Login privilege. No privilege to create database objects.
"resource"	Login privilege Privilege to create database objects
"dba"	All rights.

The concept of owner is a relationship between users and objects. Each object has one owner namely the user who created it. The owner of an object has all access privileges (see below) on it and also the right to drop the object.

The concept of access privileges regulates the right to select, insert, update, delete tuples. Select-, insert-, delete privileges are specified on table granularity , update privileges are specified on field granularity .

Each access privilege on a table for a user is either grantable or not grantable. If it is grantable , the user is permitted to forward the privilege to other users (again in grantable or not-grantable form).

All system tables are owned by the database administrator. Other users have select privilege only.

Note that the database administator can change the system tables "manually" (i.e. via `InsertStatements`, `UpdateStatements`, or `DeleteStatements`) but this may have disastrous effects and is strongly discouraged.

4.3 Multiple Connects

An application can establish multiple connections by several explicit CONNECT calls to more than one database. Retrieval as well as update queries can be issued arbitrarily against the different connected databases. Thereby, a truly distributed transaction can be performed by the application.

If a distributed transaction has performed update statements or DDL statements against more than one database, then a 2-phase commit is automatically organized thus guaranteeing consistency even in the case of a crash of one participating server machines.

Even more than one transaction can be handled simultaneously by the application. However, it is forbidden that one and the same connected database participates in more than one single transaction of that application because possible waiting conditions between those participating transactions effectively would deadlock the application.

4.4 Distributed Queries

More than one database can be addressed via a single connection by a distributed query. Table names in the FROM clause can be extended by database and machine name with the syntax: tablename@dbname@hostname. If a query exhibits a foreign database name, the kernel which has received the query for processing establishes a distributed query processing plan. Other kernels are started at demand at the addressed databases and deliver partial results to the main processing kernel. The distributed processing of the query remains transparent to the application except the explicit formulation of the remote databases within the tablenames in the FROM clause.

Distributed queries currently have the following restrictions:

DDL statements are restricted to local tables. View definitions using remote tables are not allowed.

INSERT queries on remote tables are only allowed if the table has no STATEMENT triggers (however triggers defined with the FOR EACH ROW clause are allowed). INSERT, UPDATE, DELETE statements against remote tables are possible if there are no subqueries referring to tables which are not on the same database.

SELECT queries against one remote table are not updatable, i.e. no DELETE POSITIONED or UPDATE POSITIONED is allowed.

Chapter 5

Transbase Restrictions and Limitations

This chapter gives an overview of the restrictions that apply to Transbase. Whenever possible, the restrictions are given in formulas.

5.1 On all Platforms

PAGESIZE: The size of a page is a configurable parameter, that can be set when a database is created. Changing this parameter requires databases to be archived and then restored. The default is set to: 2048 bytes

MAXDATABASESIZE: The maximum size of a database in Bytes is given by the maximum number of pages (limit is 2 power 31) multiplied with the pagesize . Thus, with a pagesize of 2 KB, the theoretical imit for the database is 4 Terabyte. Note that the absolute maximal size of the database need not be specified at database creation time. If a database runs out of space, it can be increased by adding additional database files. This is done by the database administration tool tbadmin. The initial default size for the maximum number of pages is 100000 which - in combination with the default page size - results in a default size of about 200 MB for the database. Transbase needs about one bit for each allocatable page on hard disk to keep track of allocated pages.

MAXTABLESIZE: The maximum size of a table (given in bytes) is only restricted by the (current) maximum database size.

MAXATTRNO: The number of fields in a single table is restricted to: 256

MAXROWS: The number of rows of a single table is restricted by the maximum table size. For a table with IK values, there is a restriction of 2^{31} tuples.

MAXTUPLESIZE: The size of a tuple is restricted to min(32kb-96b, PAGESIZE - 96b) bytes. See the TBX manual on how to compute the size of a given tuple. Note that the size of a BLOB is not subject to this restriction but is only restricted by the maximum file size. Within a tuple a BLOB object only contributes with about 20 byte.

MAXSTRINGSIZE: The size of an attribute of type CHAR(*) is restricted to: MAXTUPLESIZE-5.

The sum of all attributes cannot be larger than a tuple. In particular, a single attribute must not exceed MAXTUPLESIZE.

When longer attributes are needed they should be placed into BLOBs which are restriged to 2Gb.

MAXQUERYSIZE: The size of a query (i.e. its textual representation) is unrestricted.

MAXQUERYCNT: The number of concurrently active queries is restricted to 10 per each tbkernel process and per each application.

MAXCLIENTS: The number of concurrently active tbkernel processes on each database is restricted to 255. but may be less according to license restrictions. Within this range, each database specifies its own limit of concurrent users.

MAXDB: The number of concurrently active databases per each application program is restricted to 64.

MAXTA_APP: The number of concurrently open transactions per application program is restricted to 64.

MAXFROM: The number of tablenames in the FROM clause of a SELECT statement is restricted to 40.

5.2 Restrictions on Windows Platforms

5.2.1 Instantiation

By its nature, Transbase is a multi user, multiple database system with local and distributed database accesses.

The process architecture is characterized by the following property: Each application (client) is processed by a dedicated kernel (server).

In contrast to UNIX based platforms which have no restrictions with respect to number of incarnations of server processes, the different Windows platforms

exhibit many subtle restrictions and characteristics as far as the server capabilities are concerned (multitasking, shared memory, semaphors, networks, etc.).

A rough characteristic of multi instantiation of programs and runtime libraries of Windows and its derivatives is the following:

- Transbase-DLLs in general cannot be multiply instantiated However, Windows NT amnd Windows 95 automatically supports Multi Instantiation of 32-bit-DLLs, i.e. under Windows-NT/95 the 32- bit-Transbase-DLLs can multiply instantiated.
- Transbase-Executables: 16-bit-programs can only be instantiated once. 32-bit-programs can be multiply instantiated on the supported 32-bit- platforms.

The instantiation capabilities of executables and DLLs induce the following multidimensional classification:

- Number of simultaneous Transbase servers (kernels) on one machine: Single-Server/Multi-Server (SiSe/MuSe)

  As outlined above, only 32-bit database servers (tbkwso32.exe and tbkdde32.exe) on 32-bit-platforms are can act as Multi-Servers. This also induces that only this environment supports the parallel processing of several databases (by one or several applications). It also follows that only this environment supports the processing of several applications on one and the same database (on server per application).

- Number of simultaneous Transbase clients (applications) on one machine: Single-Client/Multi-Client (SiClp/MuCl)

  Multi-Clients are only possible for 32-bit-applications under Windows- NT (because DLLs must support multi-instantiation).

- Number of simultaneous database connections per application (client server connections): Single-Connection/Multi-Connection (SiCo/MuCo)

  - Multiple databases on one machine: Only for 32-bit-server auf 32-bit-platforms (cmp.. multi server)
  - Databases on different machines: Possible unless linked-in communication)

- Number of simultaneous users per database (i.e. server-datenbase cennections): Single-User-Database/Multi-User-Database

Multi-User-Database are only supported under Windows-NT with 32- bit-Servers (cmp. multi server)

Appendix C summarizes the instantiations on the different Windows platforms.

Chapter 6

Transbase System Structure

6.1 Transbase System Structure on UNIX machines

The following chapters describe the system structure of Transbase with respect to processes, system resources etc.

6.1.1 Transbase Process Structure

The Transbase process structure is shown by the following picture. LAP denote arbitrary local application programs, RAP denotes a remote (client) application program. All applications have linked in the TBX (Transbase eXec) library (either statically or linked at runtime).

When an application CONNECTs to the database at a machine M, its linked TBX library sends a message to a port of M where a Transbase demon process is listening.

The daemon process is either tbmux or tbkernel .

The daemon forks a new or activates an idle tbkernel process (denoted by TB/K1 etc. in the picture) which then serves the application. Additionally, the picture shows the Transbase Shared Memory which contains data shared between all TB/K processes.

Not shown in this diagram is another Transbase background process "tbserver" which coordinates 2-phase-commit of distributed transactions and implements asynchronous transaction abort via signals. If tbserver is not running, applications are yet able to run but 2-phase-commit and asynchronous transaction abort are not available.

Note that the names tbkernel and tbserver are often confused because one is tempted to call the daemon tbkernel a "server" process in a general sense - this

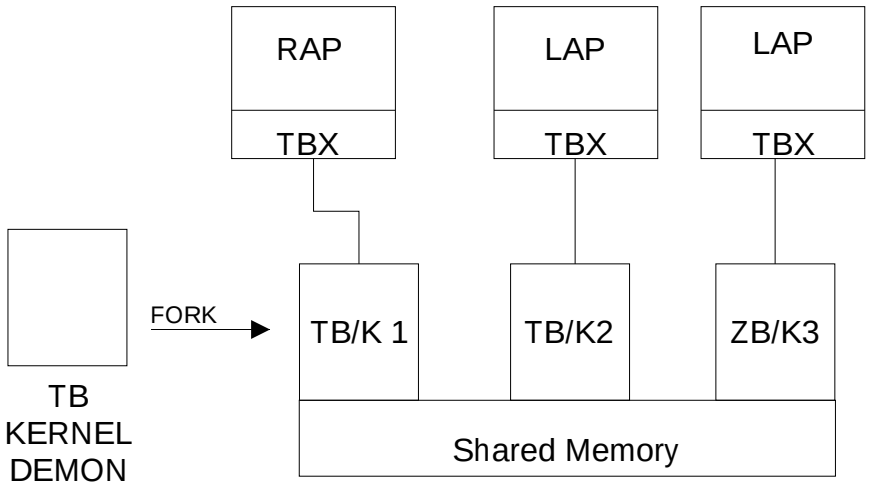

Table 6.1: Transbase Process Architecture

should have been avoided by another name for tbserver, e.g. commit coordinator "comcord" or similarily. It may be that tbserver is renamed or integrated into tbkernel in the future.

When n applications are running a transaction on one or several databases on one machine, n+1 tbkernel processes coexist in the process table of that machine. An application which is connected to the database but does not have an active transaction either has a dedicated tbkernel process or does not have one, depending on the installed daemon process: with tbkernel as daemon, a dedicated tbkernel exists, with tbmux the tbkernel processes are multiplexed between different applications (see below).

A further different process "tbadmin" exists which is an interactive tool for the database administrator to create, drop and alter databases. Its function is explained in detail in the following subsubsections which also give an overview of how the Transbase processes come into operation starting from scratch (i.e. starting with the machine's boot program).

6.1.1.1 Role of Administration Program tbadmin

With the tbadmin command "tbadmin -c dbname" a database with name dbname can be created. Many creation parameters can be given interactively or on the command line.

With the tbadmin command "tbadmin -d dbname"the database dbname is deleted.

If the database has been created as a multi-user-database, a shared memory region must be allocated for processing it. This is called booting the database. The tbadmin command ("tbadmin -b dbname") boots the database, i.e. creates the shared memory . It is possible to pass one or more database names to the tbadmin process in order to allocate a shared memory to specific databases. Omitting the database name means booting all databases on the specific machine. No access to a multi-user database is possible before the corresponding shared memory has been allocated.

A shared memory region lives from allocation until either the machine is shutdown or the explicit Transbase command "tbadmin -s dbname" is issued (where s stands for shutdown).

Note that shared memory regions even exist when no application programs are currently connected to a database. In this situation the shared memory region may be swapped out on secondary memory and thus does not require costly main memory space.

Associated with the shared memory region for a database is a set of semaphores which are used by the tbkernel processes to synchronize on the shared memory .

tbadmin also is used to change database parameters after database creation. The database is altered by "tbadmin -a dbname".

6.1.1.2 Role of Daemon Processes tbkernel and tbmux

A daemon process named tbkernel (or mykernel, depending on the license) has to be started which exports a named TCP /IP service. If such service is requested by an application program (CONNECT request), the daemon process forks a child process which exclusively serves the requesting application.

tbkernel has to be started once after the machine has been booted. All other tbkernel processes which serve the applications are started (forked) automatically from that background process whenever an application requests service.

Note that whereas the application may run under an arbitrary user identification, the actually serving tbkernel process always runs under the userid of the owner of the database directory . The owner of the database directory is the user who originally created the database.

Note further that tbkernel processes are code-shared , i.e. they are resident only once in a system (even with various databases). Of course, each tbkernel process has its own private data and stack segments.

6.1.1.3 Multiplexing with tbmux

Alternatively to tbkernel, *tbmux* can be started as daemon process. tbmux spawns tbkernel processes for applications and works as multiplexer for those spawned

tbkernel processes where the schedule basis is the transaction. The intention is to limit the number of tbkernel processes to a level which roughly corresponds to the number of applications with *active transactions*.

With this principle, the number of active connections can be much larger than the number of concurrently existing tbkernel processes. Especially, an application which has no transaction active does not necessarily occupy a tbkernel process. However, an application with an active transaction on database db has an attached tbkernel process even if the application is idle for a long time.

The degree of multiplexing can be adjusted for each database Separately by specification of mintb and maxtb, the minimum and maximum number of concurrently existing tbkernel processes *for that database*. If mintb and maxtb are both 0, then no multiplexing occurs, i.e. each application has a dedicated tbkernel process for the whole connection (it then makes no difference whether tbmux or tbkernel runs as daemon).

If mintb and maxtb are > 0 and tbmux is started then tbmux initially starts mintb tbkernel processes which all initially are in state *idle*. Whenever an application connects or starts a transaction, tbmux schedules an idle tbkernel process for that application's transaction. If no idle tbkernel exists and the number of existing tbkernel processes is less than maxtb, then tbmux starts another tbkernel which serves the application's transaction. If maxtb kernels are already running, the application is set into state *waiting* until a tbkernel becomes idle and is scheduled for that application.

Whenever a tbkernel has processed a transaction it contacts tbmux for rescheduling. It then either is rescheduled by tbmux (if an application is waiting) or - if no application requests service - is terminated by tbmux (unless number of running tbkernels decreased below mintb) or enters state *idle* until an application requests service.

Waiting applications are served on a first-come-first-serve discipline.

6.1.1.4 Role of Daemon Process tbserver

One more background process *tbserver* has to be started which exports the TCP /IP service named "tbserver". This process is responsible for asynchronous signal handling and for the commit synchronisation of distributed transactions. If none of the above services is requested, the starting of the tbserver process can be omitted.

If an application has requested one of the above services, the tbserver process forks a child that is exclusively allocated to the requesting application.

Note that though each application may have connections to more than one tbkernel processes it always has at most one connection to a tbserver process. The tbserver

process lives from the first request until the end of the requesting application or until a tbserver is requested on a different host.

6.1.1.5 Role of Library tbx.a

Each application program which wants to access a database has to link in the Transbase/Exec library tbx. This library defines a dynamic procedural interface to all database functions. By the TBX-CONNECT request an inter-process communication is initiated between the application program and the corresponding tbkernel process as described above. In the sequel, the application program (via the tbx module) and the tbkernel communicate directly over this link. However, communication details are hidden from the application programmer by the tbx module.

The tbkernel process lives until the application program explicitly DISCONNECTs from the database or until it exits or dies. A tbkernel process also may be killed by an explicit kill command. However, the permission to kill a tbkernel process is restricted to the super-user or to the database administrator.

Note that an application program may CONNECT to more than one database concurrently. In this case, separate tbkernel processes are created for each database. The application may then run distributed transactions, i.e. transactions which query more than one database but it may also be that the application simultaneously runs several transactions which all are local on one database.

WARNING: After CONNECTing to a database the application program *should not fork*. If an application program CONNECTs to a database, then forks itself and runs transactions, the attached tbkernel process gets confused about transactions and applications and probably reports errors about destroyed internal transaction tables.

6.1.1.6 Boot and Shutdown of a Database

The terms boot and shutdown of a database are directly correlated to the creation and deletion of shared memories and semaphores attached to that database, resp. After creation of a database, it is automatically booted, i.e. the shared memories and semaphores exist. A specific database db can be shut down by the tbadmin command "tbadmin -s db". This requires that no processes are running on the database (the option -sf overrides that and ungracefully stops running applications). A database can be booted by the command "tbadmin -b db". Without the parameter db, all databases on a particular machine can be shutdown or booted.

Booting a database must be (at least) done after the machine has been (re)started because a machine crash or controlled machine shutdown implicitly deletes all shared memory regions and semaphores.

Note that no explicit database shutdown is necessary before machine shutdown , because a database shutdown does not write any file to disk. If the machine is shutdown or crashes and there are active processes on databases, then all active transactions are aborted automatically and their changes are undone.

This means that with the next reboot of the database, all data is in the most recent and consistent state.

6.1.2 Role of Administration Tool wintbadm.exe

wintbadm.exe plays the part of tbadmin on Win32S and WfW32S. it is not possible to create multiuser databases.

6.2 Transbase System Structure on MS Windows

6.2.1 Role of daemon tbkdde32 and tbkwso32

tbkwso32.exe is equivalent to the tbkernel process. The main difference is, that it is automcatically add to the autostart group during installation and from then on it is started automatically each time when Windows is started.

tbkwso32.exe serves for TCP/IP connections. Since 32 Bit processes can be multiple instantiated on Win32S and WfW32S several connections to local databases are possible. Since Transbase/CD cannot handle multiuser databases on these platforms, several connects to a database are not possible.

tbkdde32.exe is the DDE equivalent of tbkwso32.exe.

6.2.2 Role of DLL tbkern32.dll

Alternatively to a client/server connection a Windows client can connect a local databases via the so called *linked-in* interface. tbkern32.dll gives the applciation the functionality of the database machine by dynamic function calls.

It is not possible to connect more than one database at a time via the linked-in interface.

Note: tbkern32.dll is loaded dynamically when the database is connected and unloaded when it is disconnected again. If an application does not terminate normally tbker32.dll will stay in main memory and unexpected errors will ocurr during the next connect. It is stricly recommended to restart Windows after such an ocurrence.

6.2.3 Role of Administration Tool wintbadm.exe

On Win32S and WfW32S wintbadm.exe plays the part of tbadmin. It is not possible to create multiuser databases.

6.3 Transbase System Structure on Windows NT and Windows 95

6.3.1 Role of 16 Bit Daemons

All 16 bit daemons can be used on Windows NT and Windows 95

Additionally there are 32 bit components of these daemons.

6.3.2 Role of daemon tbkdde32 and tbkwso32

tbkwso32.exe is equivalent to the tbkernel process. The main difference is, that it is automcatically add to the autostart group during installation and from then on it is started automatically each time when a user logges in.

tbkwso32.exe serves for TCP/IP connections. Since 32 Bit processes can be multiple instantiated on Windows NT / 95 several connections to local databases are possible.

tbkdde32.exe is the DDE equivalent of tbkwso32.exe. It serves for local and network dde connections.

6.4 Role of DLL tbkern32.dll

Alternatively to a client/server connection a Windows client can connect a local databases via the so called *linked-in* interface. tbkern32.dll gives the applciation the functionality of the database machine by dynamic function calls.

Note: It is not possible to connect more than one database at a time via the linked-in interface.

6.4.1 Role of Administration Tool wintbadm.exe

wintbadm.exe can be used adminstrate single user databases. Additionally the tbadm32.exe program is available.

6.4.2 Role of Administration Tool tbadm32.exe

On Windows NT / 95 tbadm32.exe plays the part of tbadmin.

6.4.3 Processes tbmux32.exe, tbkern32.exe and tbserv32.exe

On Windows NT/95 tbmux32 plays the part of tbmux. Additionally on Windows NT a NT service can be installed by using tbmux32.

tbkern32.exe plays the part of the tbkernel process and tbserv32.exe plays the part of the tbserver process. They are started by the tbmux32.exe process by default

6.4.3.1 Command Syntax of tbmux32.exe

Additionally to the standard parameters of tbmux, tbmux32 accepts following other parameters which must the first parameter at all.

tbmux32 install	Installs the Transbase service
tbmux32 start	Starts the Transbase service
tbmux32 stop	Stops the Transbase Service
tbmux32 pause	Pauses the Transbase Service
tbmux32 resume	Resumes the Transbase Service
tbmux32 query	Retrives info about the Transbase Service

To run tbmux32 as a console application please start:

tbmux32 noservice $parameters$

6.5 Transbase System Structure on Novell Netware

Any time when a Transbase NLM is loaded the option

```
(CLIB_OPT)/P<path to tbnlm.ini>
```

must be appended to the command line.

6.5.1 Role of Administration Tool tbadmin.nlm

On Novell netware tbadmin.nlm plays the part of tbadmin. For more information see chapter 7.1

Note: When starting tbadmin the option

```
(CLIB_OPT)/P path to tbnlm.ini
```

must be appended.

6.5.2 Processes tbmux.nlm, tbkernel.nlm and tbserver.nlm

See chapter 7.1 and 7.6 for more information.

Note: When starting any of these processes the option

```
(CLIB_OPT)/P path to tbnlm.ini
```

must be appended.

6.6 Command Syntax of tbkernel, tbmux, tbserver

6.6.1 tbkernel, tbdiag

Non-multiplexer demon

Command Line Syntax on Unix:

```
tbkernel
tbdiag
```

Command Line Syntax on Novell Netware:

```
load <path to NLM>/tbkernel.nlm (CLIB_OPT)/P<path to tbnlm.ini>
```

tbdiag is not available

File: $TRANSBASE/tbkernel or $TRANSBASE/tbdiag resp., owned by transbase (TCP version) or by root (pipe version).

The normal runtime license only comprises tbkernel. tbdiag is a separate add- on with special supervising functions.

Function: In the TCP /IP version, tbkernel (tbdiag) defines a TCP /IP service called "tbkernel" (which has to be listed in the /etc/services file). Using this service a TCP /IP port is allocated where tbkernel (tbdiag) may receive requests. Conversely, an application program uses the service "tbkernel" to identify the port where to request a database connection.

In the TCP /IP version, databases can only be accessed if a tbkernel is running.

In the TCP /IP version, if a CONNECT request is received, tbkernel forks itself; one copy (code shared) is dedicated to the requesting application program, while the other copy of tbkernel continues to offer the tbkernel service.

After the CONNECT request, tbkernel first checks if the specified database has been booted. If yes, tbkernel switches to the userid of the owner of the database directory . The name of the database directory is contained in the database file "f.db " which is read by tbkernel. If tbkernel has not been started by superuser but by another user "transbase" all tbkernel processes run under the "transbase" userid and thus only can access the database if it also has been created by "transbase".

Note: It is of some importance by which user the process tbkernel is started. If tbkernel is started by the superuser (e.g. at machine boot time), then each user "u1" can create databases (by tbadmin) that can be accessed by all other users "u2" (provided they have been explicitly granted access to the database). If tbkernel is not started by the superuser but by another user ("transbase"), then only this user "transbase" can create databases that can be accessed by all users. In this case, if a user "x1" different from "transbase" creates a database "db" with tbadmin then a user "x2" cannot access this database even if x1 has granted access rights to x2 (because process tbkernel lacks the access privileges on operating system level). Even "x1" himself can only access his database if his umask does not switch off the —xx-xx- bits.

Permissions: "-rwx——" (TCP version)

"-rwsr-sr-x" (pipe version)

Called by: Superuser or another dedicated user (called "transbase" in the sequel). Call time may be installation time of Transbase or system boot time. The pipe version is implicitly called by the application.

Environment: The TRANSBASE environment variable has to be defined and must to point to the TRANSBASE directory.

Resources: Approximate main memory requirements:

Code Segment:	900 kbytes (code-shared)
Data Segments:	300 kbytes
Stack Segment:	32 kbytes

These requirements may differ from machine to machine, depending on code generation of compilers (e.g. on RISC machines the Code Segment may be considerably larger).

Communication: TCP/IP service "tbkernel" listed in the file "/etc/services".

6.6.2 tbmux

Multiplexer demon

Command Line Syntax on Unix:

tbmux -tbk kernelprocess -tbs serverprocess
[-i *inactivity*]
[-sk *startkernel*]
[-l *stopkernel*]
[-a *spawntimeout*]

Command Line Syntax on Novell Netware:

load <path to NLM>/tbmux -tbk *kernelprocess* -tbs *serverprocess*
[-i *inactivity*]
[-sk *startkernel*]
[-l *stopkernel*]
[-a *spawntimeout*] (CLIB_OPT)/P[*pathtotbnlm.ini*]

Example (showing default options): tbmux -tbk tbkernel -i 600 -sk 5 -l 120 -a 5

Function: Starts multiplexer daemon tbmux.

-tbk kernelprocess specifies the process to be scheduled to service the application (normally this is tbkernel).

-tbs serverprocess specifies the process to be scheduled to service the application (normally this is tbserver).

Note: At least one of *-tbk, -tbs* must be specified. If one of them is not specified the corresponding server is not scheduled.

All the following options specify delay times in seconds >= 0:

-i *inactivity* specifies the time (seconds) after which an application is killed due to inactivity. This is a feature that is independent of the multiplexer technique.

-a *spawntimeout* is a timeout for the successful spawning of a tbkernel. With very heavy load of the host, the default value of 5 may have to be increased. A very high value has the slight disadvantage that detection that the partner process *tbkernelprocess* canot be started or is the wrong partner takes a long time.

-sk *startkernel* specifies the delay for the start of a new tbkernel if no idle tbkernel is found to be scheduled.

-l *stopkernel* specifies the delay for the termination of a tbkernel after it has become idle and is not needed for the start of a new transaction in the moment (and is a candidate to be killed because the number of existent tbkernel is greater than the specified minimum number).

Note that the options -sk and -l are useful to smoothen frequent termination and newstart of tbkernel processes in an environment with short transactions. With high values, the number of active tbkernel processes changes slower than with small values.

Note: For each database it can be specified whether it is subject to multiplexing or not (see command tbadmin). Of course, multiplexing only occurs if tbmux is running as daemon. With tbkernel as daemon, the muliplex specification is ignored for each database.

File: $TRANSBASE/*tbmux*, $TRANSBASE/*kernelprocess* ..

Permissions: "-rwx——"

Called by: Superuser or dedicated user transbase. Call time may be installation time of Transbase or system boot time.

Environment: The TRANSBASE environment variable has to be defined and must to point to the TRANSBASE directory.

6.7 tbserver

Command Line Syntax on Unix:

```
tbserver
```

Command Line Syntax on Novell Netware:

```
load <path to NLM>/tbserver (CLIB_OPT)/P<path to tbnlm.ini>
```

Function: tbserver coordinates 2-phase-commit for distributed transactions and transfers asynchronous abort signals to tbkernel processes.

tbserver is not needed as long as the above functions are not needed by any applications.

tbserver starts running under the userid of either superuser or transbase. In the TCP /IP version, if an application requests service, tbserver forks an incarnation of itself. In the pipe version, the application forks a copy of tbserver. The started tbserver is exclusively responsible for the requesting application.

File: $TRANSBASE/tbserver, owned by transbase (TCP version) or by root (pipe version).

Permissions: "-rwx——" (TCP version) "-rwsr-sr-x" (pipe version)

Called by: Superuser or by the user "transbase" at system boot or at installation time of Transbase. The pipe version is implicitly called by the application.

Environment: The TRANSBASE environment variable has to be defined and must to point to the TRANSBASE directory.

Communication: TCP /IP unique port

6.8 Transbase Shared Memory

As part of the tbkernel processes, a main memory region exists which is shared between all tbkernel processes which run concurrently on the same database.

In UNIX and Novell Netware implementations, the shared memory region is a system resource which exhibits the following properties:

- Access to a shared memory region is subject to the standard permission checks of the file system.
- A shared memory region is identified by a unique identifier (not a name).

- A shared memory region lives from creation until deletion. Especially, it is not deleted automatically when the last process on the shared memory region is ended. However, a shared memory region is deleted when the system is shut down.
- A shared memory region which is not in use by any process may be swapped out to secondary memory.

The shared memories are created when the database is created or booted. They are deleted when the database is deleted or shutdown . The number of shared memories depends on the database creation parameters (normally it is n+1 , where n is the number of caches specified at creation time). In most UNIX implementations, the command "ipcs -m" gives a list of existing shared memories.

Note: If shared memories are manually destroyed or if it happens that a shared memory with the key of the database exists prior to creation or boot of the database, then creation, deletion, boot or shutdown , resp., of a database may run into trouble.

Note: On Novell Netware the shared memories are owned by the NLM tbnlmipc. They are created by loading and they are destroyed by unloading this module.

6.8.1 Shared Memories on Windows NT and Windows 95

In contrast to the behaviour on Unix machines shared memories are automatically deleted by the operating systems after the last connection to a database has been closed.

6.9 Transbase Semaphores

Transbase uses a number of semaphores to synchronize parallel access of several tbkernel processes to common data.

In UNIX and Novell Netware implementations, the semaphores are subsumed under 1 semaphore vector. The semaphore vector is created when the database is created or booted. It is deleted when the database is deleted or shutdown . In most UNIX implementations, the command "ipcs -s" gives a list of existing semaphores .

Note: If semaphores are manually destroyed or if it happens that a semaphore with the key of the database exists prior to creation or boot of the database, then creation, deletion, boot or shutdown , resp., of a database may run into trouble.

Note: On Novell Netware the semaphores are owned by the NLM *tbnlmipc*. They are created by loading and they are destroyed by unloading this module.

6.9.1 Semaphores on WindowsNT and Windows95

In contrast to the behaviour on Unix machines semaphores are automatically deleted by the operating systems after the last connection to a database has been closed.

6.10 Databases and Files

Transbase employs the underlying file system to store and retrieve data from disk. Therefore, a database consists of files. In general, several files are used to store the data from the database.

A database is identified by a unique logical name which is chosen by the database administrator at creation time. The mapping between this logical name and the path name of the database directory is performed by the tbkernel process via the file f.db (see below).

If a database is to be accessed remotely, the database is identified by the pair (logical database name, host name). Thus applications may migrate from one host to another without the need for recompilation. In contrast, if a database migrates to another host, all application programs that specify an explicit database name have to be recompiled.

6.11 Runtime Environment

TRANSBASE: All Transbase processes and all Transbase application processes must have defined an environment variable named *TRANSBASE* whose contents is the pathname of the Transbase directory where all the operators and most of the auxiliary files (but of course not necessarily the databases) are stored. This directory is chosen at installation time (see the chapter "Installation of Transbase"). The definition of the TRANSBASE environment variable in each database user's profile is recommended.

TRANSBASE_LOCAL The following files of the Transbase installation are not read-only:

f.db, f.bak, f.id, and all files of directory *log*.

The above files can either be located in the TRANSBASE directory or in a separate directory whose path must be stored in a second environment variable named TRANSBASE_LOCAL.

None of the above files must be stored redundantly in both directories.

Each machine in a homogeneous network must have its own copy of the above read-write files, whereas the read-only files also can be shared via a common installation directory. This is the reason for the optional partition of the installation files into two directories.

PATH: Since the TRANSBASE directory is the location for the Transbase operators (e.g. UFI) also, it is recommended to include this path in the PATH environment variable, too.

The following examples show the environment setting for UNIX System V (under the so-called Bourne-Shell) and for Berkeley UNIX (under the C-Shell).

Example: 1

```
TRANSBASE=/usr/transbase
export TRANSBASE
PATH=\$PATH:\$TRANSBASE
```

Example: 2

```
setenv TRANSBASE /usr/transbase
set path=(\$path \$TRANSBASE)
```

6.12 Transbase, Databases and NFS

In a net of NFS (Network File System) coupled machines, each machine has to have its *private* copy of the TRANSBASE directory, i.e. the TRANSBASE directory *must not be shared* via NFS as a whole.

However, homogeneous machines can share operators, include files, text files etc. by placing them into a commonly accessible directory and using TRANSBASE variable pointing to this directory via a private path. The files f.db , f.bak, f.id, rc.Transbase, and the log subdirectory never can be shared but must be placed into separate local directories with appropriate definition of environment variable TRANSBASE_LOCAL (see Chapter Runtime Environment).

Note: It is strongly discouraged to place the data of a database on a NFS mounted filesystem. To explain this, note that when creating a database with the tbadmin command, tbadmin asks for the path name(s) of the database diskfile (s) and before image files. If a file name inside a NFS mounted filesystem were given here, then all relevant data of the database would be placed on the NFS mounted filesystem. In this situation, Transbase cannot recover the data in all situations of transaction errors or machine crashes.

This means that an *NFS mounted database can be corrupted by machine crashes* or by voluntary *killing Transbase server processes.*

Of course, the database will not be corrupted if NFS mounted diskfiles are readonly .

See also 6.13.

6.13 Disk Caches, NFS and Database Security

If properly installed, Transbase guarantees the integrity and consistency of the database even in the case of machine crashes and abnormal process ending. At transaction end, all changed database blocks are written on hard disk. If a transaction does not reach its commit point due to user's abort or a machine crash , eventually all database blocks which have been changed by the transaction are reinstalled by their copies (before images) which have been saved before the updates have taken place.

For this, Transbase uses the operation system's sync() functions. The sync function has the effect that all I/O write calls which have been issued (and perhaps are buffered in the operating system's I/O buffer) are flushed to the physical disk.

Even without the sync calls, the database would remain perfectly consistent as long as each Transbase process reaches its normal end and the host machine does not stop in an irregular way. However, in case of host machine crashes or voluntary Transbase process kills, the database's integrity heavily relies on the functioning of the database syncing.

See below how database syncing can be compromised.

NFS Mounted File Systems: When a database is placed on a file system which is NFS mounted, the database syncing is not guaranteed in most cases. In case of update activity, if either the database machine or the NFS server machine crashes, the database can be corrupted.

Note that the boot command "tbdmin -b .." issues warnings if one or several database files are placed on NFS file systems (warnings can be suppressed by the option -bf).

Disk Controller Caches: Disk controller caches may impose syncing problems which are more hidden than NFS. For the sake of increased performance, disk caches often buffer the operating system's I/O activity and sometimes even the synchronous calls thus compromising database security.

However, in many cases the *disk controlleris configurable* . For example, the cache controller may be configured in two modes, namely *write-through* or *write-deferred* (or write back or similarily). For database security, the cache must be configured in write-through mode.

Dumping and Logging: If one cannot dispense with NFS databases or with the disk speedup by write-deferring disk caches , one should take additional measurements for the database's security. Periodic dumping by the tbarc or tbtar tool is one simple possibility. Additional logging by the "Disk Recovery Logging" feature even can restore a highly up-to-date database state, however, without a correctly working sync call, some of the most recent transactions may get lost even with Disk Recovery Logging. For this topic see also chapter Transaction Recovery and Disk Recovery 11.

Read-Only Diskfiles: Of course, database security is never compromised by read operations. It follows that diskfiles moved inside a NFS file system for read-only purposes are not critical at all.

Chapter 7

Account Logging

7.1 Introduction to Transbase Account Logging

The Account Logging feature of Transbase is a general mechanism to log "Events" occurring on the database. For each event one line is added to an "Accounting-File" which holds informations such as time, event, username and also additional event-specific informations.

Examples for events which can be logged are the occurrence of errors, login attempts, the begin and end of transactions or the execution of DDL-statements. Additionally, a special event "APPLIC" is defined which can be triggered by the application via a TBX function call.

All kernels running on one database log into one database-specific file.

Events occurring at almost the same time on the same database appear in unforeseeable order in the file. All entries can be uniquely related to applications via the kernel process-id and other informations, contained in the log entry.

7.2 Layout of the Database Accounting File

For convenience, the layout of the database accounting file is such that it can be used for being spooled into a database table. It therefore has the usual Transbase external spool format (see The Data Spooler in the TB/SQL Reference Manual). For each event, one line (tuple) is appended to the accounting file.

Each line has the following entries:

- Eventtime DATETIME[YY:MS] (not null)
  the current time, when the event occurred

- Eventkey CHAR(*) (not null)
  a textual identifier, describing the actual event
- Eventinfo CHAR(*)
  additional specific information about that event
- *Kernelpid* INTEGER
  the process-ID of the associated Transbase kernel
- *Connid* INTEGER
  an identifier of the used connection
- *Username* CHAR(*)
  the database login name of the application
- *Clienthost* CHAR(*)
  the host name of the client machine where the application runs
- *Descr1* CHAR(*)
  additional event-specific description-1 as described below
- *Descr2* CHAR(*)
  additional event-specific description-2 as described below

In case of excessive length of *Descr1* or *Descr2*, the entries in the accounting file are truncated for not exceeding the maximum tuplesize.

7.3 Database Events and Associated Information

All database events, whose logging can be activated are listed below together with their specific associated informations.

ERROR The ERROR event occurs, if a Transbase kernel error happens. If the transaction is aborted due to this error, the Eventinfo is the string "Hard", otherwise the string "Soft".

Descr1 and *Descr2* hold the short- and long-error messsages, resp. as they are defined in the file "tberror.h" and are written to "*tberrtxt*".

CONN The CONN event occurs at any connect- or disconnect-call to the database. The Eventinfo is "Connected" or "Disconnecting" resp. *Descr1* and *Descr2* are both NULL.

LOGIN The LOGIN event occurs at any attempt to login to the database. The Eventinfo is "Worked" at success or "Failed" at any error. In this case, *Descr1* and *Descr2* hold the reasons for the failure ("undefined user", "wrong passwd" etc.)

TA The TA event occurs at any Begin-, Abort- or Commit-Transaction call to the database. For distributed transactions, an internal Prepare-To- Commit call is logged too. The Eventinfo is "BT", "AT", "CT" and "PT", resp. *Descr1* holds the transaction-id ("*taid*"), *Descr2* is NULL.

APPLIC The APPLIC event occurs due to a specific request from the application. See below for a detailed description of the programming interface "tbx(*SENDEVENT*, .) and mytbxsendevent(...)". The Eventinfo, *Descr1* and *Descr2* are supplied by the application.

DDL The DDL event occurs at the execution of a "Data-Definition- Language" statement.

The Eventinfo is "Dynamic". *Descr1* holds the DDL-query, *Descr2* is NULL.

DML The DML event occurs at the execution of "Data-Maipulation- Language" statements.

The Eventinfo is

Dynamic for dynamic queries

Store for STORE-statements

Openstored for OPENSTORED statements

Runstored for RUNSTORED statements.

In case of "Dynamic" and "Store", *Descr1* holds the SQL query text, otherwise it is NULL. *Descr2* is NULL in the "Dynamic" case, otherwise it holds the Statement-Id which identifies the stored query.

Chapter 8

Database Administration

The following chapters give a short overview over the database administration tools.

8.1 The Database File "f.db"

A file "f.db" located in the $TRANSBASE directory contains an internal description of all databases residing on this machine. This file is automatically updated by the administration tool "tbadmin". Although it may be inspected and even modified manually by the database administrator it is not recommended. This situation may be compared to the "/etc/passwd" file of UNIX systems which is updated automatically by tools (e.g. adduser) or may be updated manually by the super user.

Note: Each machine must have its own database file f.db. It must not be shared by different machines via NFS.

Example: An example of "f.db" is given by the following picture. This picture describes the entries for one database named "tasdb". In generally there may be many databases recorded in f.db, and one line in f.db corresponds to one database:

```
tasdb  /usr/transbase/tasdb@hp9000
hp9000  1  16425345  1
--EOF--
```

The entries are described below

position	*name*	*meaning*
1	dbname	The name of the database
2	path	The absolute pathname of the database directory
3	host	The hostname where the database resides
4	database type	for Standard DB's, entry is 1 for CD Editorial DB's, entry is 2, for CD Retrieval DB's entry is 3
5	key	A unique database identifier
6:7	resourcekeys	key of shared memory, key of semaphore, both entries 0 if database not booted
8	min kernels	if > 0: database is multiplexed with the specified minimum of kernels
9	max kernels	if > 0: database is multiplexed with the specified maximum of kernels

Table 8.1: Entries in File f.db

Explanation: The value "key" is used as default key to attach to the database shared memory and to the semaphores.

Permissions: The file is owned by the owner of the TRANSBASE directory.

Note: It is repeated here that the entries of this file should not be updated by the database administrator.

8.2 Structure of a Database Directory

Each database has its own database directory. The path name of the database directory is chosen at creation time and is denoted in f.db.

By default, all data of a database resides in files which are inside the database directory (i.e. in subdirectories of the database directory). However, at database creation time, one can also specify other locations for the database data.

Independent of the parameters chosen at database creation time, the database directory contains at least a database description file named "tbdesc". Below is an example for a tbdesc file of a Standard Database called "tasdb":

```
---
/usr/transbase/tasdb@hp9000/scr
/usr/transbase/tasdb@hp9000/drlog
/usr/transbase/tasdb@hp9000/bfims
```

```
10240  8888 19312  131072   2  10  2048  10  0 1048576
1000000  500000
/usr/transbase/tasdb@hp9000/account/acctlog
-65 1
1 0 0
TB STD SYSTEM
0 1000000 0 /usr/transbase/tasdb@hp9000/disks/tbdsk001
--EOF-
```

Now an example for a CD Editorial Database called tasdbcd:

```
/usr/transbase/tasdbcd@hp9000/roms
/usr/transbase/tasdbcd@hp9000/scr
/usr/transbase/tasdbcd@hp9000/drlog
/usr/transbase/tasdbcd@hp9000/bfims
10240  8888 19312  131072   2  10  2048  10  0 1048576 500000
10000000
/usr/transbase/tasdbcd@hp9000/account/acctlog
0 0
1 1 0
EDITORIAL SYSTEM
0 1000000 0 /usr/transbase/tasdbcd@hp9000/disks/tbdsk001
0 5000000 0 /usr/transbase/tasdbcd@hp9000/roms/cd/rfile000
--EOF-
```

The entries are described in table Entries in File tbdesc:

8.3 Space Allocation for a Database

For each database, Transbase allocates three logical data containers on non-volatile storage, namely the scratchfile container, the bfimfile container and the so called diskfiles container.

8.3.1 The Scratchfile Container

The scratchfile container is a directory where temporary files are created as needed (intermediate results, sorting, etc.) and destroyed at end of transaction. At database creation time, a path can be specified for the scratchfile directory. Transbase must have the permission to create the directory with the specified pathname. The directory can be on a foreign disk, e.g. mounted via NFS. As a default, Transbase creates the scratch directory in the database directory (as a subdirectory called "scratch").

position	*name*	*meaning*
1	rompath	The path where the romfiles are located, not used (—) in Standard Database
2	scratchpath	The path where the scratchfiles are created
3	drlogpath	Not used
4	bfimpath	The path where the before image files or logfiles (recovery) are created
5	lmsm	The size of the lock manager shared memory portion
6	smsm	The size of the segment manager shared memory portion
7	smsize	The total size of shared memory
8	local cache size	The size of the local (per-process) cache
9	ncache*size	The number of the shared memory cache segments and the size of one segment
10	number user	number of users on the database (1 means singleuser database)
11	pagesize	The pagesize in bytes
12	TA per DB	maximal number of concurrent TA's
13	not used	
14	drlogsize	maximal size of one disk recovery logfile
15	disk size	maximal size of the diskfiles in pages
16	rom size	maximal size of the rom files in pages ,not used (0) in Standard Database
17	accountpath	Path of the accounting file
18	accounting events	encoded selected accounting events
19	account on	flag, if accouning is activated or deactivated
20	diskfiles	number of diskfiles
21	romfiles	number of romfiles ,not used (0) in Standard Database
22	not used	
23	id string	TB STD SYSTEM for Standard DB's, EDITORIAL SYSTEM for CD Editorial DB's, user defined for CD Retrieval System
24	stpg	startpage of first diskfile
25	size	size of first diskfile
26	not used	
27	diskpath	path name of first diskfile
28	stpg	startpage of first romfile
29	size	size of first romfile
30	cluster	cluster number of first romfile
31	rompath	path name of first romfile

Table 8.2: Entries in File tbdesc

Although Transbase is capable of placing a database on one or several raw devices (see below), the scratchfile container always relies on the file system (i.e. cannot reside on raw device).

8.3.2 The Bfimfile Container

The bfimfile container is a directory where data is placed which serves to recover the database data after a system crash or for transaction abort ("before images"). Transbase supports two different recovery strategies namely 'before image logging' and 'delta logging' (see 11. With both methods, the changes of update transactions are temporarily logged in files which are temporarily created and dropped in the bfimfile container. By default, Transbase creates the bfim directory in the database directory (as a subdirectory called "bfims").

It is discouraged to specify the bfimfile container on NFS mounted filesystems because no guarantee for correct recovery of system crashes is given. See also 6.13.

8.3.3 The Diskfile Container

Permanent data is stored in the logical diskfile container. The diskfile container consists of 1 or several diskfiles (files of a file system) or raw devices. These diskfiles need not reside inside the same directory (but they do by default), thus the term "diskfile container" is more a logical than a physical concept.

If not specified differently, Transbase names the files tbdsk001, tbdsk002 etc. At database creation time, 1 or several diskfiles and/or raw devices can be specified. Except in the case of raw devices, Transbase must have the permission to create the specified diskfiles. As default, Transbase creates one diskfile named "tbdsk001" inside the subdirectory "disks" which itself lies in the database directory. The default size is 1000000 pages of 2KB each.

It is discouraged to specify diskfiles on NFS mounted filesystems because no guarantee for correct recovery of system crashes is given. See also 6.13.

8.3.4 Database Size and Database Extension

At creation time of the database (tbadmin command), the maximum size of each specified diskfiles must be specified. The size must be given in number of pages (blocks). As the database is filled with data, the space on disk is allocated on a demand basis, i.e. the diskfile(s) dynamically grow until their specified maximal size. If space is needed but no page on any diskfile is free, Transbase reports an error. At any time, the maximal database size can be extended by adding one or more additional diskfiles. This is done by the administration tool tbadmin. For

this purpose, the database must be shut down. Similar to database creation, each diskfile to be added can be specified by its maximal size and its placement on disk.

Of course, it has no sense to specify a diskfile size which is bigger than the operating system can provide. Transbase maintains its own space allocation scheme within each diskfile. For each allocatable page within a diskfile, there is a space overhead of one bit, roughly. For example, if a diskfile is specified with maximal size of 500000 pages (about 1 GB with 2KB pages), the space overhead and thus the initial physical size of the diskfile is about 60 KB.

Note that the first diskfile at database creation time seems to be considerably larger as displayed by the operating system. The reason is that it contains all system tables with their initial data. Another effect is that some database blocks are written at predefined addresses inside the first diskfile and leave some unallocated blocks which later on are used for user data.

8.3.5 Dedication of Diskfiles for Binary Large Objects

Transbase supports Binary Large Objects (BLOBs) as a field type of tables. Transbase also supports a mechanism to dedicate one or more diskfiles to BLOB objects. When a diskfile is dedicated for BLOBs, Transbase stores all BLOB objects (and only BLOB objects) in that diskfile.

BLOB dedication of a diskfile is specified when the diskfile is created, i.e. either at database creation time or when the database is extended by adding further diskfile(s). Note that the first diskfile can never be dedicated for BLOBs because it contains the system tables with their initial data.

If no BLOB-dedicated diskfile exists, Transbase stores BLOB and nonBLOB objects in a demand oriented interleaved manner as space is needed. Otherwise, if at least one BLOB-dedicated diskfile exists, the page allocation scheme strictly respects this. For example, if a BLOB were to be stored and the BLOB diskfiles were full whereas the nonBLOB diskfiles still had free space, Transbase would report an error and you would have to create another BLOB-dedicated diskfile.

When the first BLOB-dedicated diskfile is only created after some BLOBs have already been stored, it is no more possible to transfer the already stored BLOBs into the dedicated diskfiles except they are recreated as new objects and the old versions are dropped (via deleting and reinserting the corresponding tuples of the table).

Recall that diskfiles can be placed at different file partitions which itself reside on different media. Thus, it is for example possible to store all BLOBs in one or more dedicated diskfiles which are placed on magnettooptical disk drives (MOs). Also note that the location of diskfiles can even be changed by tbadmin after they have been created and partially or completely filled with data. This all put together makes the disk allocation scheme of Transbase highly flexible.

8.3.6 Moving Diskfiles after Database Creation

The location of some or all of the existing diskfiles of a database can be changed after creation of the database. The database alter command "tbadmin -a dbname" is used for this purpose. There the new locations are given interactively and tbadmin uses the operating system's copy command to move the diskfile.

It is also possible to copy diskfiles "by hand" and to adapt the pathnames of the corresponding files in the database description file "tbdesc". Note, however, that it is not guaranteed that this procedure will also be supported in future versions of Transbase.

Note furthermore that it is not possible to split or merge diskfiles by hand.

8.3.7 Read-only Diskfiles

When data is inserted, updated or deleted by SQL commands or some DDL command is performed, one or several diskfiles are physically updated by Transbase. In this case, the Transbase process needs write access to the corresponding diskfile(s).

When a session starts by the applications's CONNECT command, the attached Transbase database process "tbkernel" always opens the first diskfile for reading access of some system blocks and to check the user's login password etc. Further diskfiles are opened on a demand driven basis depending on the data being requested.

Transbase always tries to open a diskfile in read/write mode. If this does not succeed, the diskfile is opened in read-only mode. This means that database processing is even possible if one or several diskfiles have been moved to file systems which are mounted read-only or even to read-only media such as CD-ROM. Of course, modification of data on read-only diskfiles is not possible.

Recall that diskfiles can be dedicated to BLOB objects. Thus it is e.g. possible to store all BLOB objects on low cost media such as magnetooptical disks or even transfer them to CD-ROM media. This may be an interesting cost effective database partitioning which physically reflects the different data characteristics (read-only versus read/write, or high traffic versus low traffic).

8.3.8 Jukebox Support

A Transbase database can be fully or partially stored on a jukebox. This can be arranged at database creation time or by successively moving database diskfile(s) to the jukebox system. Some new aspects are introduced by the characteristics of some jukebox drivers. When Transbase tries to open a diskfile which is not online,

the jukebox system possibly must change a disk in one of its disk drive. Perhaps the driver requires that all files of the disk to be set offline have to be closed.

Transbase provides for some commands to explicitly control the closing of the diskfiles. In principle, the file closing commands define events when to close diskfiles (e.g. after each query or after fetching a BLOB) and limit the number of parallel open files. The commands can be issued by the application and thus are explained in detail in the TB/SQL Reference Manual.

8.3.9 The Romfile Container

Romfile containers have no meaning for Standard Databases. Their meaning for CD Editorial and CD Retrieval Databases is explained in the Transbase CD-ROM Database Guide.

8.4 TBADMIN Command Line Options

The utility tbadmin provides several administrative operations for databases. Its first parameter selects one of the following operations (shown when tbadmin is called without parameters):

-b boot database

-s shutdown database

-r reboot database

-i inform about database

-c create database

-a alter database

-d delete database

-C attach to CD-ROM database

-M migrate database

-drec disk recovery

Each command option has further parameters that can be seen interactively by the command

```
tbadmin params option
```

For all command options the flag -nv can be specified which suppresses any version output.

In the following, each command option is described in detail.

8.4.1 tbadmin -b

This tbadmin option boots databases, i.e. recovers them from previous crashes and installs their corresponding shared memories. A database must be booted before it can be accessed by programs.

```
tbadmin -b[f] [dbname ...]
```

The optional f flag ignores potential semaphore collisions and omits NFS check on files

Valid Parameters are: database names of those databases to be booted. Without further parameters, all databases on the local machine are booted.

Note:

- A database may be booted multiply.
- Booting a database may be time consuming when the database had crashed and a lengthy transaction had been active that must be rolled back.
- Especially under UNIX, tbadmin must have sufficient privileges to create semaphores and shared memories and to change their ownership into the owner of the database files.
- Version conflicts can prevent a database from being booted. In particular, when a database has been created with an old Transbase version, it must be migrated to a new version. See tbadmin -M below.
- The database cannot be restored from a previous crash. In that case, a disk recovery process must be started manually.

8.4.2 tbadmin -s

This tbadmin option shuts databases down, i.e. saves their buffers persistently to disk and uninstalls their corresponding shared memories and semaphores. After shutdown the database cannot be accessed by programs.

```
tbadmin -s[f] [dbname ...]
```

The optional f flag terminates all active Transbase kernel processes thereby aborting any active transactions on the database.

Valid Parameters are: database names of those databases to be shut down. Without further parameters, all databases on the local machine are shut down.

Note:

- A database may be shut down multiply.
- Booting a database may be time consuming when the database had crashed and a lengthy transaction had been active that must be rolled back.

Potential Errors:

- Without the f flag, an error is returned when Transbase kernel processes are running on the database.

8.4.3 tbadmin -r

This tbadmin option is equivalent to shut down and reboot databases.

```
tbadmin -r[f] [dbname ...]
```

8.4.4 tbadmin -i

This tbadmin option retrieves information about all databases (no parameters) or about a specific database (dbname parameter).

```
tbadmin -i [dbname [parameters]]
```

Without dbname parameter, a list of all database names on the local host is given. For a specific database (specified by its dbname), either all available information is displayed or a specific information is retrieved specified by the following parameter list:

n	Database Name
h	Database Home (Path of Database)
nd	Number of Database Disk Volumes
d	Database Disk Volume Paths
ds	Database Disk Volume Sizes in Pages
nr	Number of Database CD-ROMs
r	Database Rom Path
rs	Database Rom Volume Sizes in Pages

vid Database Identification

bf Database Bfim Path

s Database Scratch Path

drs Database Logging File Size in MB

dr Database Disk Recovery Path

drl Database Disk Recovery Logging

acc Database Accounting (file,events,status)

u Number of Users

min Number of min. multiplexed Kernels

max Number of max. multiplexed Kernels

ps Page Size in Bytes

lc Sorter Cache Size in KB

nc Number of data cache areas with <size> KB

st Database Status

cp Database Codepage

loc Database Locale

typ Database DB Type

ta Number TAs per Database

id Database Ident

f.db Database f.db Entry in following syntactic sequence: (n,h,host,sy,id,st)

tbdesc Database Description Path

connections Connected Kernels to that Database

connections=hostname Connected Kernels to that Database from a specific client which can be identified either by hostname or IP address.

locks Lock Situation on that Database

space Shows the number of occupied and free pages for each disk file.

Example: The command `tbadmin -i h` can be used in SHELL scripts to access database files:

```
hp=‘tbadmin -i dbname h‘
cp $hp/disks/* $tape
```

To list all connected clients on a database:

```
tbadmin -i dbname connections
```

8.4.5 tbadmin -c

This tbadmin option creates a new database on the local machine. There is an interactive way `tbadmin -c dbname` and a non-interactive, batch-oriented way `tbadmin -cf dbname param ....` Please note that the dbname parameter is mandatory. The params that may be specified in the non-interactive method are listed below:

`typ=S|E` Database Type: Standard(S) or CD-Editorial(E)

`rom=<path>` Database rompath: a valid pathname where the romfiles may be found. If omitted, defaults to rom/cd.

`h=<path>` set database home. If omitted the database home directory will be located in the current directory and named `dbname@hostname`

`d=[<path>] [, [<size >] [, [blob] [,pinit]]]`
single database disk file specification (needed for every disk file)

`<path>` must be a valid path name. If omitted the pathname defaults to disks/ tbdsk001 in the database home directory.

`<size>` specifies the maximum size of disk file in pages. If omitted, `<size>` defaults to 102400 pages.

`<blob>` dedicates the disk file for storage of blobs. If omitted, defaults to FALSE.

`<pinit>` forces the diskfile to be physically initialised (with empty pages). If omitted, defaults to FALSE.

`bf=<path>`
must be a valid path name for the Before Image directory. If omitted, the pathname defaults to bfim/ in the database home directory.

`s=<path>` must be a valid path name for the scratch directory. If omitted, the pathname defaults to scratch/ in the database home directory.

`drl=on|off` If set to on, disk recovery is switched on for the database, i.e. logfiles will be kept even if they are no longer needed for transaction recovery. If omitted, disk recovery defaults to off.

`drs=<size in MB>` specifies the size of logfiles in MBytes. If set to 0, database is switched to Before-Image-Logging.

`rs=<total rom space>` specifies the number of database pages reserved for ROM space. For details, see the CD-ROM guide.

`acf=<path>` must be a valid path name for the accounting file.

`ace=[<evkey>[,<evkey>]...]` Lists the events to be logged into the accounting file. Valid event keys are: NONE, ALL, ERROR, CONN, LOGIN, TA, APPLIC, DDL, DML, STATQ, STATS

`acl=on|off` Switches Database Accounting on or off. Events to be recorded are kept when switch off, so that they can be reactivated.

`ci=on|off` If on, the database will be case-insensitive, i.e. all identifiers will be mapped to upper-case. If omitted, case-insensitivity defaults to off.

`p=<password>` specifies the tbadmin password. If omitted, the tbadmin password is the empty string.

`u=<number>` sets the maximum number of concurrent sessions on the database. It cannot be set higher than permitted by the Transbase server license. u=1 set the database into single-user mode.

`min=<number>` sets the minimum number of tbkernel processes controlled by tbmux

`max=<number>` sets the maximum number of tbkernel processes controlled by tbmux

`ps=<size in B or KB>` sets the page size for the database. `<size>` must be one of: 1k, 2k, 4k, 8k, 16k, 32k or 64k. If omitted, page size defaults to 2k. See Tuning Guide

`lc=<size in KB>` sets the size of the local (sorter) cache which is allocated for each database instance. If omitted, the local cache size defaults to 512k. See Tuning Guide

`nc=<number>[*<size in KB>]` specify number and size of the shared memory areas allocated for the database. If `<size>` is omitted, it defaults to 64k. If the option is completely omitted, one shared memory of size 128k is allocated by default. See Tuning Guide

`tbdesc=<path> f.db=<path>` create a database with settings identical to those referenced by the f.db and tbdesc files.

`loc=<locale setting>` specifies the "locale" for the database. It must be a valid string on the host operating system, e.g. `en_GB.ISO8859-15` or `de.UTF-8`. If omitted, `<locale setting>` defaults to "C".

`cp=euc|jis|ascii|propriet|utf8|singlebyte` sets the character string coding of the database; valid values are: propriet, ascii, si for single-byte codings and utf8, euc, jis for multi-byte codings. If omitted, the coding defaults to propriet.

`crypt=yes|no` This option enables database encryption. The default is `no`.

The encryption mode is a creation parameter and cannot be changed later.

`mt=max|det|min|off` configures multithreading behaviour. Setting this option to `max` activates the full potential of multithreading; it establishes data pipelines in query plans that run in parallel, also using out-of-order execution, for improved overall performance. The setting `det` offers fair parallelism while producing result sets in deterministic output order. Performance is likely to suffer somewhat compared to maximum parallelism, as data pipelines operate only in first-in, first out mode. The setting `min` uses a rather defensive strategy of parallel query execution; parallel execution is limited to I/O relevant nodes (e.g. REL or REMOTE) and activates work-ahead for the entire SELECT query. Finally `off` means no parallelization at all. This is also the default.

Note:

- The default settings of memory sizes are extremely low for backward compatibility.
- The default setting of the page size is very low; in any case, it should be set higher than the block size of the underlying file system which is typically 4k or even 8k.
- Some of the settings above cannot be changed after the database has been created, while others can be changed by tbadmin -a.
- The physical initialization of disk files may take time and put IO load on the machine.

8.4.6 tbadmin -a

This tbadmin option modifies database parameters after the database has been operational. The database is shut down and rebooted after the given parameters have been changed.

Note that some database parameters like the page size cannot be changed by tbadmin -a.

There is an interactive way `tbadmin -a dbname` and a non-interactive, batch-oriented way `tbadmin -af dbname params`. Please note that the dbname parameter is mandatory.

The parameters that may be specified in the non-interactive method are listed below:

`n=<dbname>`
The database is renamed to `<dbname>`.

`dx=[<path>] [, [<size >] [, [blob] [,pinit]]]`
The database is extended for one disk file with specified properties. Multiple dx options are allowed to extend the database for more than one diskfile.

`<path>` must be a valid path name. If omitted the pathname defaults to disks/ tbdsk00n in the database home directory.

`<size>` specifies the maximum size of disk file in pages. If omitted, `<size>` defaults to 102400 pages.

`<blob>` dedicates the disk file for storage of blobs. If omitted, defaults to FALSE.

`<pinit>` forces the diskfile to be physically initialised (with empty pages). If omitted, defaults to FALSE.

`bf=<path>` must be a valid path name for the Before Image directory.

`s=<path>` must be a valid path name for the scratch directory.

`drl=on|off` switches disk recovery logging on or off. If on, logfiles will be kept for archiving even if they are no longer needed for transaction recovery. If off, logfiles are automatically dropped when no longer needed.

`drs=<size in MB>` specifies the size of logfiles in MBytes. If set to 0, database is switched to Before-Image-Logging. If set to a value >0, database is switched to Delta-Logging. See Tuning Guide

`acf=<path>` sets the `<path>` for the accounting file.

`ace=[<evkey>[,<evkey>]...]` sets the events to be logged into the accounting file. Valid event keys are: NONE, ALL, ERROR, CONN, LOGIN, TA, APPLIC, DDL, DML, STATQ, STATS

`acl=on|off` switches Database Accounting on or off. Events to be recorded are kept when switched off, so that they can be reactivated.

`ci=on` Switches the database to be case-insensitive thereby mapping all identifiers to to upper-case. An error occurs, if identifier names collide. Note that a database cannot be switched back to case-sensitivity.

`u=<number>` sets the maximum number of concurrent sessions on the database. It cannot be set higher than permitted by the Transbase server license. u=1 sets the database into single-user mode.

`min=<number>` sets the minimum number of tbkernel processes controlled by tb-mux

`max=<number>` sets the maximum number of tbkernel processes controlled by tbmux

`lc=<size in KB>` sets the size of the local (sorter) cache which is allocated for each database instance. See Tuning Guide

`nc=<number>[*<size in KB>]` specifies number and size of the shared memory areas allocated for the database. If `<size>` is omitted, it defaults to 64k. See Tuning Guide

`loc=<locale setting>` specifies the "locale" for the database. It must be a valid string on the host operating system, e.g. `en_GB.ISO8859-15` or `de.UTF-8`. If omitted, `<locale setting>` defaults to "C".

`mt=max|det|min|off` configures multithreading behaviour. Setting this option to `max` activates the full potential of multithreading; it establishes data pipelines in query plans that run in parallel, also using out-of-order execution, for improved overall performance. The setting `det` offers fair parallelism while producing result sets in deterministic output order. Performance is likely to suffer somewhat compared to maximum parallelism, as data pipelines operate only in first-in, first out mode. The setting `min` uses a rather defensive strategy of parallel query execution; parallel execution is limited to I/O relevant nodes (e.g. REL or REMOTE) and activates work-ahead for the entire SELECT query. Finally `off` means no parallelization at all. This is also the default.

Note:

- The physical initialization of disk files may take time and put IO load on the machine.

8.4.7 tbadmin -d

This tbadmin option deletes a database. The database is shut down first and then all files are deleted and the database entry in f.db is deleted.

```
tbadmin -d[f][nv] dbname [p=<password>]
```

If the database has a non-empty tbbadmin password, the password can be specified as command-line-option (f flag) or it is prompted interactively. Please note that the dbname parameter is mandatory.

8.4.8 tbadmin -C

This option controls the "attach" operation which creates a database from a CD-ROM image. See CD-ROM guide. Command-line-parameters:

`tbadmin -C[f|F][nv] dbname [<parameter> ...]`

If the f flag is specified, no interaction occurs except for CD-ROM insertion. If the F flag is specified, no interaction occurs at all. In these cases, parameters have to be specified at command-line or be set to their default values as specified below.

`h=<path>` Database Home: a valid pathname where the database's home directory will be placed. If omitted the home directory will be located in the current directory and named "dbname@hostname"

`d=[<path>] [, [<size >] [, [blob] [,pinit]]]` Database disk file specification (needed for every disk file). `<path>` must be a valid path name. If omitted the pathname defaults to disks/tbdsk001 in the database home directory. `<size>` specifies the maximum size of disk file in pages. If omitted, `<size>` defaults to 102400 pages. `<blob>` dedicates the disk file for storage of blobs. If omitted, defaults to FALSE. `<pinit>` forces the diskfile to be physically initialised (with empty pages). If omitted, defaults to FALSE.

`bf=<path>` must be a valid path name for the Before Image directory. If omitted, the pathname defaults to bfim/ in the database home directory.

`s=<path>` must be a valid path name for the scratch directory. If omitted, the pathname defaults to scratch/ in the database home directory.

`r=<path>` must be a valid path name for the directory where the Romfiles are located (mounted).

`rf=<file>` must be a valid file name for of a Romfile.

`drl=on|off` If set to on, disk recovery is switched on for the database, i.e. logfiles will be kept even if they are no longer needed for transaction recovery. If omitted, disk recovery defaults to off.

`drs=<size in MB>` Specifies the size of logfiles in MBytes. If set to 0, database is switched to Before-Image-Logging.

`acf=<path>` must be a valid path name for the accounting file.

`ace=[<evkey>[,<evkey>]...]` Lists the events to be logged into the accounting file. Valid event keys are: NONE, ALL, ERROR, CONN, LOGIN, TA, APPLIC, DDL, DML, STATQ, STATS

`acl=on|off` Switches Database Accounting on or off. Events to be recorded are kept when switch off, so that they can be reactivated.

`p=<password>` specifies the tbadmin password. If omitted, the tbadmin password is the empty string.

`u=<number>` sets the maximum number of concurrent sessions on the database. It cannot be set higher than permitted by the Transbase server license. u=1 set the database into single-user mode.

`min=<number>` sets the minimum number of tbkernel processes controlled by tbmux

`max=<number> <number>` sets the maximum number of tbkernel processes controlled by tbmux

`lc=<size in KB>` sets the size of the local (sorter) cache which is allocated for each database instance. If omitted, the local cache size defaults to 512k. See Tuning Guide

`nc=<number>[*<size in KB>]` specify number and size of the shared memory areas allocated for the database. If `<size>` is omitted, it defaults to 64k. If the option is completely omitted, one shared memory of size 128k is allocated by default. See Tuning Guide

`loc=<locale setting>` specifies the "locale" for the database. It must be a valid string on the host operating system, e.g. `en_GB.ISO8859-15` or `de.UTF-8`. If omitted, `<locale setting>` defaults to "C".

`cp=euc|jis|ascii|propriet|utf8|singlebyte` This option sets the character string coding of the database; valid values are: propriet, ascii, si for single-byte codings and utf8, euc, jis for multi-byte codings. If omitted, the coding defaults to propriet.

`mt=max|det|min|off` configures multithreading behaviour. Setting this option to `max` activates the full potential of multithreading; it establishes data pipelines in query plans that run in parallel, also using out-of-order execution, for improved overall performance. The setting `det` offers fair parallelism while producing result sets in deterministic output order. Performance is likely to suffer somewhat compared to maximum parallelism, as data pipelines operate only in first-in, first out mode. The setting `min` uses a rather defensive strategy of parallel query execution; parallel execution is limited to I/O relevant nodes (e.g. REL or REMOTE) and activates work-ahead for the entire SELECT query. Finally `off` means no parallelization at all. This is also the default.

8.4.9 tbadmin -M

This tbadmin option is used to migrate a set of databases that has been created with an older Transbase version to the current version.

For versions older than V5.3 you must specify a codepage for each subset of databases. The codepage specification is ignored for databases of version 5.3 and younger.

```
tbadmin -M [[dbname]+ [cp=euc|jis|ascii|propriet|utf8|singlebyte]]+
```

Note:

- This procedure is not reversible. Before migration, it is strongly recommended to back up the database using a backup tool such as tbarc (of the original software version).

8.4.10 tbadmin -drec

This tbadmin option is used to dump and restore databases completely. This option requires the database to be delta-logged (`drs>0`). While a database is being dumped, it may yet be operational ("fuzzy dump"), i.e. no shutdown is required and the database can process user queries while being dumped.

To create an initial dump of a database for later differential dumping use the command:

```
tbadmin -drec dump db=<dbname> dir=<targetdir> [sh=[<shellcmd>]]
```

where `sh=<shellcmd>` is a shell command to be carried out after every disk-file/logfile was dumped.

Create consecutive differential dumps for an initial dump with the command:

```
tbadmin -drec dump diff[erential] db=<dbname>
          dir=<targetdir> [sh=[<shellcmd>]]
```

To dump a database into a single file or directly onto tape, use one of the previous commands and replace `dir=<targetdir>` with `file=<file>|<device>`, e.g.,

```
tbadmin -drec dump db=<dbname> file=<file>|<device>
```

If no target directory or file is supplied, the dump will be written to stdout.

```
tbadmin -drec dump db=<dbname>
```

Note: When dumping to a stream, i.e. to a file, device or stdout, the size of each file to be dumped must not exceed 4GB.

To extract dumped files of a database from a single file or tape, use the command:

```
tbadmin -drec extract file=<file>|<device> [dir=<dir>]
```

This will create the same structure in `<dir>` as if the dump was originally written into that directory.

If the optional `dir=<dir>` is omitted, then the files are restored into their original location from where they have been dumped.

The command

```
tbadmin -drec inspect file=<file>|<device>
```

produces a list of database files disk/logfiles from the given file or device. Note that for the dumping procedure is lenghly, only a preliminary file list will be extracted from the file header and not the complete stream is processed. Thus the file might also contain additional files that were created while the dumping process was running and some *.act files might have become *.arc files before being dumped.

`tbadmin -drec extract` only copies files into the database directory. It does not start any recovery actions, yet.

To restore a database from a dump, use the command:

```
tbadmin -drec restore [reboot] db=<dbname>
          {dir=<dir>|file=<file>|<device>}
```

If `<dir>` contains an initial dump and a database `<dbname>` does not exist on the system, then a new database `<dbname>` is created, using the dumped database configuration settings. The user is prompted for database settings that may be changed during the recovery process, e.g. pathnames. The command line option `-drecf` forces that all interactive prompts are omitted, and the dumped configuration is used unchanged. If `<dbname>` already exists, then recovery is performed in-place and existing files are overwritten. If `<dir>` contains no initial dump, then `<dbname>` must exist. Recovery from differential dumps will copy all diskfiles from the dump, but will rather apply logfiles directly to the diskfiles where possible, instead of copying, and thus reduces the required disk space of the recovery process. This procedure may also be carried out iteratively with changing `<dir>` names or its contents, resp., e.g. if files have to be extracted from tapes or file-archives. Note, that if the database is to be recovered from file or device, then the stream will be extracted automatically to your your disk, i.e. diskfiles are copied directly into the database directory. Logfiles are extracted into a temporary directory in your database home directory, since random read access to logfiles is required during recovery which is not possible from a stream.

The disk recovery can be restricted to restore the database only until a given point in time (e.g. when an erroneous transaction shall be backed out). Use the command:

```
tbadmin -drec restore <dbname> [YYYY.MO.DD.HH.MM.SS]
```

to specify a date value. In this case, no commits later than the specified date will be restored.

To determine whether a dump is complete and what period a dump covers, use the command:

```
tbadmin -drec verify dir=<dir>
```

The output will list all relevant files in the dump and supplies for logfiles the timestamp of the last commit. This information might prove useful for recovery until a given point in time. An error occurs if the dump is not complete, i.e. if information is missing between diskfiles (or first logfile, resp.) and last logfile in dump. If only diskfiles and no logfies are available in `<dir>`, no information on commit timestamps is available. This command is only applicable on dumps residing in a directory on disk, not on dumps in a file archive or on tape.

The recovery procedure is completed by rebooting the database, where all open transactions are rolled back producing a consistent database state. The optional `restore reboot` command for the very last recovery step, allows to restarts the database as soon as restoration is completed and brings the database in operational state.

```
tbadmin -drec restore reboot <dbname> [YYYY.MO.DD.HH.MM.SS]
```

8.5 Account Logging Administration

The account logging component is maintained using the administration tool "tbadmin" or "myadmin" resp. (see *Transbase System and Installation Guide*). The following examples use the "tbadmin" call, options and parameters are identical with "myadmin".

The account logging settings can be examined by calling

Syntax on Unix:

```
tbadmin -i <database> acc
```

Syntax on Microsoft Windows:

```
tbadm32 [-notify task msg | -notifywindow wnd msg] -i
<database> acc
```

Syntax on Novell Netware:

```
load <path to NLM>/tbadmin -i <database> acc
          (CLIB OPT) /P<path to tbnlm.ini>
```

8.5.1 The Account Logging File

By default the account logging file of a database has the name "acctlog" and is located in the database subdirectory "account". Name and location of this file can be choosen at database creation time either interactively or by use of the command line parameter "acf=`<path>`".

The accounting file can be changed via `tbadmin -a[f] ... [acf=<path>] ....` This has the effect that the old file is closed and a new, empty file will be created if account logging is switched on.

Note that the old accounting file is not deleted. It can be analyzed or spooled into the database for further evaluation.

8.5.2 Switching Account Logging On and Off

Account logging can be switched on or off at creation time or later on using "`tbadmin -a[f]` ...". This can be performed with the command line parameter "`acl=on|off`" or also interactively.

Note that the accounting status (`on` or `off`) is remembered during the shutdown-boot phases of the database. Therefore account logging is automatically resumed after boot if it was switched on at shutdown.

8.5.3 Activating and Deactivating Accounting Events

The set of events to be logged can be specified or changed via "tbadmin" either interactively or by use of the command line parameter "ace=`<evset>`".

Hereby `<evset>` is an optional sequence of absolute settings, followed by an optional sequence of incremental settings:

```
<evset> :: [<evkey>]... [+|-<evkey>]...
<evkey> :: ALL|NONE|ERROR|CONN|LOGIN|TA|APPLIC|DDL|DML
```

Event keywords (`<evkey>`) are case insensitive, separators are any of `<blank>`, `<tab>`,',' and ':'.

If the specified set starts with an absolute setting, the set of active events is overwritten. If the specified set starts with an incremental setting, the set of active events is changed.

Example: The table below shows the effects of `ace` settings.

ace=error,ta	set to ERROR and TA
ace=+ddl, -ta	add DDL, remove TA
ace=all-error	set to all events except ERROR
ace=none	switch all events off
ace=-all	switch all events off

8.6 Code Pages and Locale Setting

From version 5.4 on, Transbase supports code pages and Locale settings that can be configured dynamically, when creating a database. Neither the code page nor the Locale setting can be modified after the database has been created.

The code page determines how character strings are coded. The Locale setting determines what characters are considered characters and non-characters as well as how characters are mapped to uppercase and lower case.

The Locale setting is not used to determine the sort order, i.e. the sort order is always given by the machine code of the underlying computer.

8.6.1 Code Pages

Transbase supports the following code pages:

Proprietary

: This code page should be used when a database created by a former version of Transbase shall be accessed. No assumptions about coding are made. Each character is one byte wide. Characters of different code schemata can be mixed arbitrarily.

SingleByte

: This code page is very similar to Proprietary, except that character functions respect the Locale setting of the database.

ASCII

: This code page restricts all character strings to consist of ASCII characters (below 128) exclusively. Please note that runtime errors may occur when BINCHAR is assigned to CHAR.

UTF-8

: This code page provides full Unicode support. Internally (and at the communication layer) each Unicode (UCS-2) character is coded by a variable

sequence of one, two or three bytes. Each UCS-4 unicode character is coded by up to six consecutive bytes.

In UTF-8 databases, the assumption that a character takes one byte may be false. Character functions in Transbase work (and count) in term of characters (not bytes). Space allocation functions, however, still work (and count) in bytes.

8.6.2 Locale Setting

Former versions used the so-called C-locale setting which is the default for all C programs. From version 5.4, a specific locale can be specified for a database that is used for all Transbase kernels on that database.

The Locale setting has influence of the character classification and also on the mapping from upper case to lower case and vice versa (i.e. on the case sensitivity).

Please note that a reasonable Locale setting is even needed for UTF-8 databases in order to map e.g. German Umlaut characters correctly. If no Locale setting (or the C Locale setting) is used, such characters would not be mapped.

8.7 Case Sensitivity in Transbase Databases

8.7.1 Principles

Each Transbase database either works in case sensitive (CS)or case insensitive (CI) mode. Affected are all identifiers existing in the database schema as are names of tables, views, fields, sequences, integrity constraints. Names are created by DDL statements, e.g. by CREATE TABLE, CREATE INDEX etc. Names are referenced for example in SELECT statements and are then resolved in either CS or CI mode. In CS databases, names are stored in the catalogue tables as they have been created. Name resolution succeeds if the referencing name exactly matches the name as stored in the catalogue. In CI databases, all names are stored in upper case letters in the catalogue tables (except names given in delimiter identifiers, see next section). Name resolution converts the name to be resolved to upper case and then tries to match it against the stored names.

For example, a table has been created via `CREATE TABLE suppliers ...`

In a CS database, referencing the table in a SELECT statement requires the formulation `SELECT * FROM suppliers ...`

In a CI database, one could also formulate: `SELECT * FROM SUPPLIERS ..` or also `SELECT * FROM SuPplIeRs ...`

š54 Please note that the upper case mapping of non-ASCII letters is determined by the Locale setting of the database.

8.7.2 Delimiter Identifiers in CI databases

In a CI database, all names created by DDL statements are stored in upper case in the catalogue tables except names which have been formulated in double quotes (so called delimiter identifiers). The latter are stored exactly as they have been written (of course without the quotes). For example, `CREATE TABLE "suppliers" ..` inserts the table name suppliers (in lower case). The statement `SELECT * FROM suppliers ..` then fails to be resolved because unquoted names are converted to upper case first in a CI database. The table can only be referenced by a quoted name again, as in: `SELECT * FROM "suppliers" ...`

8.7.3 Migration from Case Sensitive to Case Insensitive

A database created as case sensitive can later be migrated to a CI database provided that some conditions on the already existing object names are fulfilled. The migration can be made within the usual interactive configuration procedure "tbadmin -a .." . However, it might be detected that the set of already existing names cannot be converted to UPPER case in the catalogue tables without collisions. In this case the migration is rejected with an error. Note that queries in existing applications might fail to run after migration but only if names are referenced with delimiter identifiers. All other queries run without change after the migration.

8.8 Using tbadmin on MS Windows

Syntax:

```
tbadm32 [-notify task msg | -notifywindow wnd msg] <other
options>
```

Effect: This option is used when tbadmin is started from another task to perform a specific action. Especially during setup it is very useful to attach or create databases using tbadmin.

The -notify/-notifywindow options makes tbadmin to send his exitcode to *task/wnd* by using the message msg. The exitcode will be the wParam component of *msg*. tbadmin will close his window at the end of the operation silently.

Chapter 9

TBADMIN Library Interface

9.1 General

The programs require to include file "tbadmsdk.h". The TBADM function library is only available for Windows.

All functions resemble exactly the corresponding functions of the tbadmin command. In particular, parameters are usually passed to the library as strings. If there is no specific argument, the parameter can be set to NULL.

The library functions return TRUE(1) to signal no error and FALSE(0) otherwise. In case of errors, the functions TbadmGetLastErrorCode and TbadmGetLastErrorText can be called for details.

9.2 TbadmLoad

Syntax:

```
TbadmLoad();
```

TbadmUnLoad

Syntax: TbadmUnLoad();

TbadmAttach

Syntax: TbadmAttach(char *dbname, char *args, PTbadmCallback ptr)

char *dbname: Name of the database

char *args: Arguments, see tbadmin command

PTbadmCallback ptr: Pointer to a callback function.
The function is called for every CD to attach and has to return TRUE on success. It may display details to the user, e.g. rpath or label to allow him to supply the correct CD. A typical look of AttachCallback may be:

```
Bool AttachCallback
(char *dbname, char *rpath, int rno, char *label)
{
    int ret = MessageBox( (HWND)0, dbname, label, MB_OKCANCEL);
    if (ret == IDCANCEL) return FALSE;
    else return TRUE;
}
```

9.3 TbadmCreate

Syntax:

```
TbadmCreate(char *dbname, char *args)
```

`char *dbname`: Name of the database `char *args`: Arguments, see tbadmin command, e.g. "ps=4096 nc=2*1024"

9.4 TbadmAlter

Syntax:

```
TbadmAlter(char *dbname, char *args)
```

`char *dbname`: Name of the database `char *args`: Arguments, see tbadmin command, e.g. "nc=4*1024" or "drs=0"

9.5 TbadmDelete

Syntax:

```
TbadmDelete(char *dbname, char *args)
```

`char *dbname`: Name of the database `char *args`: Arguments, see tbadmin command.

9.6 TbadmBoot

TbadmBoot(char *dbname, Bool forced) char *dbname: Name of the database Bool forced: FALSE(0) ... standard boot, TRUE(1) boot forced; see tbadmin command.

9.7 TbadmShutdown

Syntax:

```
TbadmShutdown(char *dbname, Bool forced)
```

`char *dbname`: Name of the database

`Bool forced`: FALSE(0) ... standard shutdown, TRUE(1) forced shutdown; see tbadmin command

9.8 TbadmGetInfo

Syntax:

```
TbadmGetInfo(char *dbname, char *args, TbadmInfo **dbinfo)
```

`char *dbname`: Name of the database or NULL

`char *args`: Arguments or NULL

`TbadmInfo **info`: pointer to a structure pointer of type TbadmInfo to receive information TbadmGetInfo returns information like the command "tbadmin -i". However, a structure of type TbadmInfo is used for transferring the results programmatically.

In case dbname is not specified (NULL), only a list of valid database names is provided in dbinfo->dbnamelist. The list is a `\0`-separated list of names which is ended by an empty string. The following code fragment shows how to display this list:

```
char *db = info->dbnamelist;
while (db && *db) {
    printf("%s\n", db);
    db = db + strlen(db) + 1;
}
```

TbadmGetInfo returns information about a specific database if the dbname argument is specified. The information returned is delivered in the fields of the structure of type TbadmInfo.

The following code fragment shows how to display information about a specific database:

```
printf("%s\n", dbinfo->dbentry.d_name);
for(i=0; i<dbinfo->volumes.nrvol;i++)
printf("%-60.60s %ld\n",
dbinfo->volumes.tbadmvolume[i].volpath,
dbinfo->volumes.tbadmvolume[i].size);
```

9.9 TbadmGetLastErrorCode

Syntax:

```
TbadmGetLastErrorCode(long *e);
```

`long *e`: Pointer to long receiving the error code.

9.10 TbadmGetLastErrorText

Syntax:

```
TbadmGetLastErrorText(char **etxt);
```

`char **etxt`: Pointer to character string pointer receiving the pointer to the error text.

9.11 TbadmMigrate

Syntax:

```
TbadmMigrate(char *dbname)
```

`char *dbname`: Name of the database

The result of TbadmMigrate is analogous to the command tbadmin -M.

9.12 TbadmMonitor

Syntax:

```
TbadmMonitor(char *dbname, char *args, PTbadmMonitorCallback ptr)
```

`char *dbname`: Name of the database or NULL

`char *args`: Arguments or NULL

`PTbadmMonitorCallback ptr`: pointer to a callback function that is called for delivering the monitor results.

The signature of the callback function and its possible layout is shown in the code fragment below:

```
Bool MonitorCallback(TB_Statistic *tbstat, int *res)
{
    printf("%ld\n", tbstat->act_time);
}
```

Chapter 10

Tools

10.1 Archiving, Restoring, Schema Report

The commands tbtar and tbarc serve to archive Transbase databases in ASCII form (text files) and to recreate databases by reading the produced output.

Archives in text form can be used to create *backup copies* of database states as well as for database migration between machines which are not binary compatible. In summary:

tbarc: produces or loads a collection of spool files. Each table is represented by one file (plus additional files for BLOB values). A tbi-like script *make.db* serves to control the loading process. This is a rather flexible form of a Transbase archive, e.g. it is possible to modify the scripts (in a limited way) before the database is rebuilt.

tbarc is also able to quickly produce the *make.db* script without the spool files thus generating a schema overview of a database (the script creation time is independent of the table sizes).

tbtar: can write the database data together with the schema information into one single output stream or into one or several files. By tbtar, also very big tables can be handled whose file size of tbarc would exceed the maximum file size supported by the underlying OS.

tbtar also can restrict the output to one specific table thus producing a "table archive" which can be loaded onto a new or existing table.

To write a database onto tape , the tbtar stream can be piped to the *tbtape* command which can write a stream onto one or several tapes. To read back a database from tape, the reverse procedure can be applied, i.e. tbtape pipes the tape stream into tbtar which builds a new database. Thereby the intermediate storage on disk needed by tbtar can be configured by the "ds" option of tbtar.

10.1.1 The tbarc Command

Syntax on Unix:

```
tbarc   -w  archive   dbname [ p= passwd ] [ c= cldef ]

tbarc   -r  archive   dbname

tbarc   -s  archive   dbname
```

Syntax on Windows:

```
tbarc32   -w  archive   dbname [ p= passwd ] [ c= cldef ]

tbarc32   -r  archive   dbname

tbarc32   -s  archive   dbname
```

Syntax on Novell Netware:

```
load <path to NLM>/tbarc -w  archive dbname [ p= passwd ]
          [ c= cldef ] (CLIBOPT)/P<path to tbnlm.ini>

load <path to NLM>/tbarc -r  archive dbname
           (CLIBOPT)/Ppath to tbnlm.ini

load <path to NLM>/tbarc -s  archive dbname
           (CLIBOPT)/Ppath to tbnlm.ini
```

Effect: *tbarc* -w archives a database by spooling all database data and by producing a script *make.db* for the restoration of the database. All files are placed into one directory given by parameter *archive* . As far as possible, the names of the spool files are the same as the table names, otherwise synthetic names are generated. BLOB values are represented by file names referring to subdirectories which contain one file for each BLOB value.

make.db contains CREATE statements for tables, indexes and views as well as SPOOL statements from files to tables.

For *tbarc -w*, the specified archive is created if it does not exist, otherwise it must be empty.

tbarc -s produces *make.db* without spool statements and without producing spool files for the tables. This option runs very fast and produces a complete schema description of a database (without archiving the database).

tbarc -r restores the database by applying an archive script (as made by *tbarc -w*) to a (presumably) empty database.

Options for both tbarc and tbtar are:

cldef : The specification of parameter cldef supports the cluster partitioning of CD-ROM databases. Details are explained in the *Transbase CD-ROM Database Guide*.

p=: tbarc /tbtar run under the database userid of "tbadmin". Without the p= option, the caller is prompted for the appropriate password.

Limitations: Although a tbarc archive also preserves all information about domains, constraints, referential constraints and privileges, it is not possible to edit *make.db* with respect to these table properties. This is because *make.db* does not contain corresponding DDL statements but catalog table data in spool format. See the *rse* command of tbi to change a table w.r.t. constraints.

make.db is divided into three parts: it first handles all domains. Part 2 contains DDL statements for the creation of tables and indexes and data loading.Part 3 contains DELETE and UPDATE statements on catalog tables to reinstall privileges and constraints. Additional table definitions in Part 2 can be added, but arbitrary modifications in Part 2 in general make Part 3 inconsistent and should be avoided.

Examples for tbarc: Writing a db to a tbarc archive :

```
tbarc   -w dbarchive db p=root
```

Restoring the database:

```
tbarc   -r dbarchive db
```

Producing a schema description:

```
tbarc   -s dbarchive db
```

10.1.2 The tbtar Commands

Syntax on Unix:

```
tbtar   -w      dbname  options ...

tbtar     -r|-rl|-ro     dbname  options ...
```

Syntax on Microsoft Windows:

```
tbtar32   -w       dbname  options ...

tbtar32     -r|-rl|-ro     dbname  options ...
```

Syntax on Novell Netware:

not available

Effect: *tbtar -w* writes one data stream on stdout or file. The stream embeds all data of a database (DBarchive) or all data of one single table (tablearchive). tbtar -r reads an input stream produced by *tbtar -w* and stores the data into the target database.

tbtar -rl is like *tbtar -r* but uses the local SPOOL variant (see SpoolTableStatement *in the Transbase SQL ReferenceManual*). This variant can be used if the tbtar process and the Transbase kernel process run on the same machine. This speeds up the restoration and saves intermediate disk space.

tbtar ro reads an input stream produced by *tbtar -w* without building up the database. This can be used as a procedure to test the physical correctness of a tbtar archive.

Options for -w:

`p=[<passwd>]`

tbadmin *passwd*

`f=outfile [,maxsizeMB]`

write to file(s) instead stdout, maxsizeMB is the (optional) maximum size of the outfile in MB.When a specified maximum is reached, tbtar switches to the next specified output file if there is one specified, otherwise tbtar stops. This option can be multiply specified.

`st=<tablename>`

tbtar writes a "tablearchive" which only contains data of one table.

`skip=<number>`

applicable to st= option: skip `number` tuples of the table

`count=<number>`

applicable to st= option: stop after writing `count` tuples of the table

`c=<cldef_file>`

see usage of "tbarc"

Options for -r:

`p=[<passwd>]`

tbadmin password

`f=<infile>`

read data from one or several file(s) instead of stdin. This option can be multiply specified.

`ds=<tmpdiskspaceMB>`

This option limits the size of the intermediate spoolfiles which serve as an intermediate data container of tbtar's input data. Without this option, the size of each intermediate spoolfile is proportional to the size of the corresponding table, thus disk overflow may occur. Small file sizes, however, slow down the loading process. The default file size is 1GB.

`nf=<numberblobsperloadportion>`

Loading a table with Blob fields requires one file per blob in a specific directory. This option limits the number of files by breaking up the input stream into several portions (similar to the ds= option). The default value is 1000.

`tt=<tablename>`

applicable to tablearchive only: load a tablearchive to an existing targettable. With this option, all CREATE statements in the archive are ignored. See Limitations below.

`rsfile=<outfile>`

for tablearchive only: all schema statements occurring in the input stream are stored into `file`.

Limitations: In contrast to a DBarchive, loading a tablearchive may fail because a table is not an isolated object. A table may use a domain, it may reference other tables via referential constraints or via view definitions. If the target environment does not satisfy one of these conditions, the loading process fails. The rsfile= option may help to extract enough information to adapt the target database to the conditions required by the source table.

Examples for tbtar:

- Writing the database onto stdout:

  tbtar -w db

- Writing the database onto two files of maximal 100 MB each:

  tbtar -w db f=arch1,100 f=arch2,100

- Restoring the database from two files:

  tbtar -r db f=arch1 f=arch2

- Restoring the database with limitation of the temporary diskspace to 2 MB:

  tbtar -r db ds=2 `dbarchivefile`

- Restoring a database from a file "archive" with limitation of the number of blobs spooled in one step to 100000 and limitation of intermediate file size to 500 MB:

  tbtar -r db nf=100000 ds=500 f=archive

- Restoring a local database using fast spool variant :

  tbtar -rl db f=archive

10.1.3 The tbtape Command

Syntax on Unix:

`tbtape -w [-Bsize] device`

`tbtape -r device Version tb418 and lower`

`tbtape -r [-c] device Version tb419 and higher`

Syntax on Windows:

not available

Syntax on Novell Netware:

not available

Effect: tbtape-w writes its input stream to the specified device or file. tbtape writes in block sizes of `size` KB where 5KB is the default. If the output medium (tape , floppy) is full, tbtape requests for a further medium before continuing.

Tbtape writes sequence numbers on the first block of each physical medium to check the correct medium sequence during reading. On each block a continuation flag is written to recognize whether the file has a continuation on a *further medium* instance.

tbtape -r reads a tbtape file from the specified device and copies it to standard output. If the medium is exhausted but the tbtape file is not finished then tbtape prompts for another medium instance.

The -c option can be optionally used for binary compatibility between different machines. Using the -c option makes tbtape converting integers in the block headers to the machine representation. When the medium cannot read, tbtape assumes a binary incompatibility and asks you to use the -c option and to retry.

Example: Writing the database on one or several tape (s):

```
tbtar   -w db | tbtape   -w -B5  /dev/rst0
```

Restoring the database from tape (s), available diskspace 2 MB:

```
tbtape   -r /dev/rst0 | tbtar   -r  db  ds=2
```

10.2 Tbcheck - Correctness Test and Statistics of a Database

Syntax on Unix

`tbcheck [ options ]` *`dbname`* `[` *`tbadminpassword`* `]`

Syntax on Windows

`wintbchk [ options ]` *`dbname`* `[` *`tbadminpassword`* `]`

Syntax on Novell Netware:

```
load <path to NLM>/tbcheck [ options ]  dbname
     [  tbadminpassword ] (CLIB OPT)/P<path to tbnlm.ini>
```

Effect: tbcheck inspects the database *dbname* and produces on stdout a report about all tables, indexes, BLOBs and BLOB indexes. The report comprises the number of occupied pages on all B-tree levels, the number of tuples and the average occupancy of pages and other info.

At the same time, tbcheck performs an intensive check of all data structures used for storage of database data.

For this command the database must be shutdown.

Options:

-s *tname* Restrict the check and report on the table tname and its indexes.

-T*number* Print number tuples of each table.

-d print a dot '.' after 20 pages checked.

Example:

```
tbcheck  tasdb
tbcheck -T100 -s bigtable tasdb passwd
```

10.3 Tbdiff - Difference between two Databases

Syntax on Unix

```
tbdiff [-v] scriptdir  db1  db2
```

Syntax on Windows

```
tbdiff32 [-v] scriptdir  db1  db2
```

Syntax on Novell Netware:

not available

Effect: tbdiff records all differences between two databases db1 and db2.

tbdiff produces a script and possibly some spoolfiles in directory "scriptdir". The script is named "diff.sql" and is suitable for tbi to be run.

When run on db1, "diff.sql" transforms db1 to the state of db2.

tbdiff runs under tbadmin but otherwise appears to db1 and db2 as normal application. tbdiff leaves both databases unchanged. However, intermediate changes are made on db2 which are reset before tbdiff exits.

tbdiff prompts for tbadmin to log in.

The "v" option protocols the actions on screen.

The directory scriptdir must either exist and be empty or must be creatable by tbdiff XE "tbdiff" .

Example:

```
tbdiff  -v scriptdir  tasdb1 tasdb2
```

10.4 Cleaning Log Records for Distributed Transactions

Syntax on Unix

```
tbrecord -c  logfiles ...
```

Syntax on MSWindows and Windows for Workgroups and WIN32S

not available

Syntax on Windows NT / Windows 95

```
tbrec32 -c  logfiles ...
```

Syntax on Novell Netware:

```
load <path to NLM>/tbrecord -c  logfiles ...
          (CLIB OPT)/P<path to tbnlm.ini>
```

Effect: tbrecord -c cleans logfiles in the "log" subdirectory of the TRANSBASE directory.

Explanation: For distributed transactions which make updates on more than 1 database, there is a small file written in the subdirectory "log" of the TRANSBASE directory of one of the participating machines. This file serves to coordinate the atomic distributed commit processing. It has a name like "logxxx" where xxx is a number and contains a so called log record which indicates whether the transaction has reached its commit point. Normally, this file is automatically deleted at the end of the application. However, if the application crashes, the logfile is maintained until all participating database kernels have finished the corresponding transaction. tbrecord is the utility which decides if a logfile is still needed and which deletes it if it is no more needed.

Note: The only reason for this command is to delete useless logfiles from time to time. For the correctness and atomicity of transactions it is not mandatory to use this command. Beside that, it is rather unlikely that many superfluous logfiles are produced.

Files: Files located in the "log" subdirectory within the TRANSBASE directory.

Example:

```
tbrecord -c $TRANSBASE/log/log*
```

Chapter 11

Transaction Recovery and Disk Recovery

11.1 Transaction Recovery

Transaction recovery deals with the event that an update transaction is not committed but has to be rolled back such that none of its effect remain in the database. Different circumstances may lead to the rollback of a transaction, namely either an explicit abort call of the application or a hard runtime error detected by the kernel such as lack of dynamic memory. The corresponding kernel itself then performs the rollback.

In case of an unexpected kernel exit or a machine crash, possibly several transactions may remain uncomitted and must be rolled back either by other kernel instances or by the database booting procedure after reboot of the machine.

Transbase supports two different transaction recovery methods called 'before image logging' and 'delta logging'. The choice is a database configuration parameter which can be altered via tbadmin (the latter requires a shutdown and reboot of the database).

11.1.1 Before Image Logging

With this method, before images of changed database pages are stored in a file 'bfmxxxxx' (bfims directory) which is shared among all current transactions. To roll back a transaction, the corresponding pages are written back from the before image file into the database diskfiles. The before image file is deleted when no update transaction is active. Sometimes this is deferred to avoid frequent creation and deletion, but at least at database shutdown time it is finally deleted.

This recovery method is well suited for long update transactions, especially for mass loading, index creation and database build up. However, it performs bad for

short update transactions with heavy load, because all changed pages are forced to the diskfiles at commit time followed by a sync call to the (usually very large) diskfiles.

After a system crash, all committed transactions are already reflected in the diskfiles, thus only the interupted transactions have to be rolled back by the same mechanism as described for runtime rollback.

11.1.2 Delta Logging

Delta logging writes a sequential stream of database changes into so called 'logfiles' which are files L0000000.act and L0000001.act and so on (each having identical size which is configurable). The changes consist of records which describe the updates on database pages on byte level. At commit time, the produced logfile entries in memory are forced to their corresponding logfile. Changed pages may remain in the database buffer pool and will later be asynchronously written to diskfile when the page is no longer needed. While new logfiles L*.act are added, older logfiles eventually are deleted or (if disk recovery is switched on, see later) are renamed to the extension .arc. To roll back a transaction the corresponding logfile entries are read backwards and applied to the pages.

After a system crash, the existing logfiles are used for two purposes: Already committed transactions whose effects are not yet completely on diskfiles are redone and interrupted transactions are made undone on diskfiles. This requires reading the logfiles in both directions.

After a database shutdown, at least the most recent logfile remains existent because it is not deleted until it is filled up to its specified length.

11.2 Disk Recovery with Delta Logging

Disk Recovery is a method to protect the most current database state against loss of the diskfiles.

Note that only delta logging (in contrast to before image logging) provides disk recovery functionality. Additionally, the disk recovery option has to be specified with tbadmin to provide for disk recovery.

Note that Disk Recovery and Archiving are related closely, see "Archiving, Restoring, Schema Report" 10. With Archiving, a snapshot of the database is made at a certain point in time. Restoring an archive reinstalls that database state and thus in general does not reinstall the current database state.

In contrast, disk recovery provides the possibility for a backup of the most recent database state in case that one or more diskfiles are destroyed.

To enable disk recovery, the database must be switched to delta logging and the disk recovery log must be switched on. Detailed commands will be given in the following section.

For disk recovery, the database must be dumped periodically with a special tbadmin option. A database shutdown is not necessary for this dump. The database state reflected by a dump perfomed while the database is operational, lies somewhere during the dumping procedure, therefore this dump method is sometimes called a "fuzzy dump". The generated dump must be kept on a save place, e.g. a tape or at least on another harddisk.

11.2.1 Full Dump

A full dump consists of all diskfiles and of all L*.act logfiles at the time of dumping. There may be older logfiles L*.arc at that time in the bfims directory, i.e. they already have been renamed from .act to .arc, but they do not become part of the new dump. They logically belong to an older dump if there exists one.

When a full dump has been made, all changes since that dump are logged into further logfiles in the bfims directory. Together with the dump, they provide for the information to construct the current database state after diskfiles have been lost.

All logfiles produced after the dump must be moved periodically to a save place. Logfiles L*.arc need not remain in the bfims directory, but note that the newest logfiles L*.act must not be moved from the bfims directory before they have become L*.arc.

11.2.2 Differential Dump

Alternatively to the Full Dump a differential dump method is provided. For differential dumping an initial dump, very similar to a full dump, is required. Afterwards only logfiles changes since the last dump are moved to a save place, making current L*.arc files obsolete and thereby cleaning up the bfims directory of the database.

Recovery with differential dumps requires the initial dump, and all subsequent differential dumps (consisting of L*.arc and L*.act files) and, if available, all logfiles from the database's bfims directory. During recovery process logfiles are directly applied to the diskfiles where possible, reducing disk space consumption during the recovery process.

11.3 Commands for Disk Recovery

11.3.1 Database Configuration for Disk Recovery

The following is for setting the database into the state required for disk recovery.

Within "tbadmin -a db" specify:

```
Disk Recovery Log on [y|n]  >> y
Logging File Size in MB [ >= 0 ] >> 10
```

The latter option sets logfile size to 10 MB. Any value greater than 0 automatically swichtes to delta logging.

11.3.2 Dump Commands

The following command produces an initial dump into a directory 'dumpdir':

```
tbadmin -drec dump db=<dbname> dir=<dumpdir>
```

Whereas a differential dump is made with:

```
tbadmin -drec dump diff[erential] db=<dbname> dir=<dumpdir>
```

Instead of dumping to a directory, a dump into one single file 'file' is made by replacing `dir=<dumpdir>` with `file=<file>` in the dump commands, e.g.

```
tbadmin -drec dump db=<dbname> file=<file>
```

On UNIX platforms, the dump can also be written onto a sequential drive:

```
tbadmin -drec dump db=<dbname> file=/dev/rmt1
```

Finally, the dump is written to stdout, if no target directory or file is supplied.

```
tbadmin -drec dump db=<dbname>
```

Note: When dumping to a stream, i.e. to a file, device or stdout, the size of each file to be dumped must not exceed 4GB.

11.3.3 Rebuilding a Database from a Differential Dump

We assume that a database db is destroyed.

Step1: Save all the remaining logfiles which are in the bfims directory of db. Add them to the other already saved logfiles.

Step2: Make the initial dump online accessible in a directory dumpdir on a local or a mounted disk drive. Direct recovery from a dump that resides in a single file or on a peripheral device is possible. But it can also be extracted into a directory dumpdir by:

```
tbadmin -drec extract file=<file>|<device>  dir=<dumpdir>
```

Step3a: Restore an existing database in-place.

There are two ways to do that: If the directory structure of the destroyed db is intact, then in-place recovery is possible.

```
tbadmin -drec restore db=<db>
         {dir=<dumpdir>|file={<file>|<device>}}
```

This command will use the database setting as provided in the configuration file residing in `<db>`'s database directory. First all diskfiles are overwritten with their older copies from the initial dump. Afterwards the bfims directory is cleaned. Remember that logfiles have been saved in Step1. Finally all logfiles from the initial dump are copied to the bfims directory. The database now is in its initial state as it was when the first dump was made.

Step3b: Build a new database.

If the directory structure of the destroyed db is not intact, or the database is to be recovered to a different location, then

```
tbadmin -drec restore db=<new_db>
         {dir=<dumpdir>|file={<file>|<device>}}
```

will build a new database `<db_new>` in the current directory, using the configuration file from the dump. During the creation process various database configuration settings (like paths) are changeable. The `-drecf` command forces database recreation without prompting for optional configuration settings, using all settings provided by the dumped database configuration.

Repeat the command you have chosen (Step3a or Step3b) until all differential dumps have been applied.

Step4: If logfiles of the crashed database were saved in Step1 then apply them, too, with the command from Step3.

Step5: Finally, after the last set of logfiles has been processed, issue:

```
tbadmin -drec reboot db=<db>
```

resp.

```
tbadmin -drec restore reboot db=<db>
```

if the database is to be restarted immediatly after the last dump is applied.

The database then is at its most recent state reflected by the saved logfiles and can be processed again.

Chapter 12

Installation of Transbase

This chapter describes how the superuser or database administrator brings the Transbase database system into operation.

In the first section, the system resources that are needed, are described, in the second section the installation procedure is given, and in the final section the runtime environment is described.

12.1 Installation under UNIX

12.1.1 System Resources

12.1.2 Installation Procedure

Media: Transbase is delivered either on tape or on diskettes. In either case, Transbase is delivered in cpio format.

Reading: "cd" into the directory where you want to install Transbase. Extract the files using:

```
% cpio -idumvBc <drive
```

where drive denotes your appropriate device.

After this command a new directory exists, the so called TRANSBASE directory (usually you will find its name to be "tb")

If you use a csh you now type:

```
% setenv TRANSBASE /usr/transbase/tb
```

Alternatively with a Bourne Shell you would type:

f.id	f.db	sqlca.h?
tberror.h	tbfid.h	tbx.h
tbx.a		
tbkernel	tbserver	tbrecord
makeper		
tbperm	ufi	tbarc
tbadmin	tbi	rc.Transbase
tbpp	tbp1	tbp2
log	dat	optree
util	dat/*	optree/*
util/*.hlp		

```
$ TRANSBASE = /usr/transbase/tb
$ export TRANSBASE
```

Finally you must type:

```
% sh $TRANSBASE/tbperm
```

Note that this last tbperm command must be started under superuser (tbperm performs "chown root .." commands for program objects where the s-bit is set). Note additionally that the PATH variable must be such that chown can be found by tbperm. Finally, be prepared to be asked for license numbers by tbperm interactively.

Files: After the above commands have been run, the following files have to exist in the TRANSBASE directory. Note that depending on license restrictions some of the given files may be missing or some additional files will be present.

SHARED MEMORY: Transbase needs a number of shared memories per database. By default, the number of shared memories is three, namely a shared memory for shared kernel data plus two shared memories for pages of user tables "cache"). The number of cache shared memories can be changed from 1 to a machine dependent maximum. The size of each shared memory is 64 kbytes or less.

SEMAPHORES: Associated with each database is a set of four semaphores which are used to synchronize on the shared memories. The superuser has to arrange for an appropriate system environment by setting particular system constants and possibly reconfiguring the operating system. The following constant settings are recommended:

Those constants are contained e.g. in the files /etc/conf/modules/shm/space.c and /etc/conf/modules/sem/space.c (System V).

For BSD 4.2 they are located in /sys/h/shm.h and /sys/h/sem.h

SHMSEG	6
SHMMIN	1
SHMMAX	64 kbytes
SEMMNI	10
SEMMNS	60
SEMUME	10

TCP/IP: Communication software must be installed and the appropriate daemons must be running.

Disk Space: Transbase installation needs disk space of about 6 Mbytes (this may vary considerably on different machines). Not included here is the space needed for databases.

Userid: It is recommended to install a UNIX userid for the maintenance of Transbase. This userid is assumed to be "transbase" in the following, the "transbase" home directory is assumed to be given by the environment variable TRANSBASE.

12.1.3 Runtime Environment

/etc/services: In the TCP/IP version, two entries have to be made in the file /etc/services as follows:

```
tbserver    2024/tcp    # comments
tbkernel    2025/tcp    # comments
```

The first field defines the name of the service, namely "transbase" or "tbkernel". The next field defines that the services can be requested via TCP/IP port number, e.g. 2024. The port number given here has to uniquely identify this service in your local environment. If a numbers less than 1024 are chosen, only superuser-processes can listen on the specified port; since tbkernel and tbserver may be started by superuser, a port number setting of `< 1024` would guarantee that only those processes and no user process could listen on that port number.

tbserver: In the TCP/IP version, the process tbserver has to be started. Its runtime environment variable "TRANSBASE" must have been set to the location of the Transbase operators. Starting of tbserver must be done after each machine reboot or after the process has unexpectedly stopped for whatever reason.

tbkernel: In the TCP/IP version, the process tbkernel has to be started. Its runtime environment variable "TRANSBASE" must have been set to the location of the Transbase operators. Starting of tbkernel must be done after each machine reboot or after the process has unexpectedly stopped for whatever reason.

tbadmin -b: Before accessing a database, it must have been "booted". Booting the database must be done after the machine has been booted or after the database has been explicitly shutdown. The TRANSBASE environment variable has to be defined.

application: The application must have the TRANSBASE environment variable defined. In order to access Transbase tools, it is recommended to include TRANSBASE in the PATH environment also.

Note: A script rc.Transbase located in the TRANSBASE directory may be used to start processes with appropriate environment setting and to boot databases. "rc.Transbase" is produced by calling `$TRANSBASE/tbperm`.

Note: For the sake of database security, it is recommended to pay special attention to 6.13.

12.1.4 Boot Script

The setting of the runtime environment is recommended to be done from the boot script. Examples are given below for System V and BSD 4.2.

The simplest way to do all database startup is to call the shell script "rc.Transbase" located within the TRANSBASE directory.

Example:

```
if [ -f $TRANSBASE/rc.Transbase ] ; then
    sh $TRANSBASE/rc.Transbase
fi
```

System V `File /etc/rc3.d/S15transbase:`

```
TRANSBASE=/usr/transbase
export TRANSBASE
$TRANSBASE/tbkernel
$TRANSBASE/tbserver
$TRANSBASE/tbadmin -b <boot list>
```

This file is executed automatically, when the system switches to system state 3.

BSD 4.2 The file /etc/rc should contain the following lines:

```
TRANSBASE=/usr/transbase
export TRANSBASE
$TRANSBASE/tbkernel
$TRANSBASE/tbserver
$TRANSBASE/tbadmin -b <boot list>
```

12.2 Installation under Windows

Transbase is delivered on several installation disks (3 1/2 "[EL1]). The installation procedure is the same for Windows 16 Bit and Windows NT Windows 95. The installation is started by execution of SETUP.EXE (Disk 1) from the file manager or from the program manager. After a short initialization phase the dialogue oriented installation and configuration is performed.

The following parameters are requested by the installation:

- Installation-Directory Needed disk memory depends on the chosen volume of packages and varies from about 2 MByte until 15 Mbyte.
- Installation package(s) and corresponding license numbers.
- Type of client server communication (Network Driver)

The installation performs the following:

- Copying (with decompression) of the Transbase packages into the installation directory.
- Activation of the packages by patching licence numbers into specific files.
- Creation of file TBWIN.INI in the installation directory. If another tbwin.ini is found in the Windows directory, this will be copied to the installation directory first.
- Creation of a program group with Transbase symbols.

After installation, program SETUP.EXE can repeatedly be called from the Transbase installation directory to change or adapt package licence numbers or communication protocol.

12.2.1 Configuration

Some configuration parameters for Transbase as well as for Windows and the network are crucial for the effective and sound processing of Transbase.

12.2.1.1 Localisation of Parameters

Transbase uses the following hierarchical search for its configurations parameters:

TBWIN.INI Transbase specific initialisation file in the Transbase directory.

TBWIN.INI Values which have not been found in step 1 are searched in file tbwin.ini in the Windows directory (often `C:\WINDOWS`).

WIN.INI - Section TBWIN Values which have not been found in step 1 or 2 are searched in the Windows file WIN.INI (Windows installation directory) in the section [TBWIN].

DOS-Environment-Variables Finally, values which have not been found in step 1 - 3 are searched in the DOS environment variables (normally defined via AUTOEXEC.BAT).

Note: The most important environment component of Transbase is the variable TRANSBASE.

DOS based Transbase tools (e.g.ESQL preprocessor) do neither resolve via TBWIN.INI nor via WIN.INI. This means that the TRANSBASE variable must be defined as DOS environment variable (possibly beside additional entries in TBWIN.INI or WIN.INI).

12.2.1.2 Transbase Parameters

At installation time, the file TBWIN.INI is created with corresponding entries. Its parameters can be changed either by a reconfiguration (call of SETUP.EXE in the installation directory) or manually with a normal text editor (e.g. notepad.exe). Note that section name as well as parameter name are case insensitive, however the parameter value is case-sensitive!.

An example of file TBWIN.INI is shown below, together with a short explanation of parameters. A more detailed explanation is given in the corresponding chapters.

```
[TBWIN]
TRANSBASE=D:\TB    // Transbase Installation Directory

// communication protocol:
HOSTACCESS16=tbwsoc16.dll    // for 16-bit environment
HOSTACCESS32=tbbun32.dll    // for 32-bit environment

// local DB access
```

```
LINKED_IN=ON
// ON: local databases are connected via linked_in DLL
// OFF: local databases are connected via client server

// Overriding service names (of file SERVICES)
// refers to TCP/IP communication
TRANSBASE_SERVICENAMES=tbkernel:tbserver

// alternatively: direct definition of port numbers
;TRANSBASE_SERVICENAMES=2024:2025

TBKERNEL=2024    // only relevant for old PC/TCP versions
TBSERVER=2025    // (compatibility)

EDITOR=notepad    // only relevant for ISQL  tool WINTBI.EXE:
// editor started by WINTBI command 'e[dit]'
```

Note: The following parameters are mandatory:

- TRANSBASE
- HOSTACESS16 or. HOSTACCESS32
- All other parameters are optional.
- If the variable EDITOR is not defined, the WINTBI command 'e[dit]' has no effect.
- Unless the service names for network communications are directly defined via the `TRANSBASE_SERVICENAMES` variable, the file SERVICES are searched for entries of the form:
- tbkernel `<port>/<protocol>`
- tbserver `<port>/<protocol>`
- The default value of variable `LINKED_IN` is OFF.

Note: If in a reconfiguration the host access variant `Linked_In` is chosen the entry `LINKED_IN=ON` is automatically generated.

A further reconfiguration, however, does not set back this entry.

12.2.1.3 Operating System

There are no Transbase configuration parameters which refer to the specific operating system.

Conventional memory in DOS-based Windows environments is treated like in other normal programs: Transbase does not directly use DOS memory but only indirectly via the underlying network protocol.

Note: When Windows for Workgroups is running and an additional network protocol (e.g. `PC/NFS`) is started, problems in network access often indicate a lack of conventional memory. This is not a Transbase specific problem.

12.2.1.4 Communication

Linked-In For a `linked_in` communication between client (application) and server, no special configurations parameters are necessary.

TCP/IP TCP/IP protocols access entries in the following files:

- File HOSTS

  This file holds a mapping between symbolic host machine names and Ethernet addresses.

  A Transbase client connects to a databse by supporting a database name in the form:

  ```
  <database>@<host>
  ```

  The hostname can either be a symbolic host name (which then is resolved via the file HOSTS) or directly the ethernet address (e.g. 192.9.200.9).

- File SERVICES

  Via this file, all available network services are listed and their corresponding port addresses are defined. The following entries are relevant for Transbase:

  ```
      ...
  tbkernel    <tbk_port>/tcp
  tbserver    <tbs_port>/tcp
      ...
  ```

  These services must have been inserted both on client and server machine with consistent port addresses.

  The variable `TRANSBASE_SERVICENAMES` (file TBWIN.INI or WIN.INI) can override the above service names or directly define port addresses.

DDE Requires no special configuration.

DECnet Only the Transbase services with port addresses must be defined

12.3 Installation on Novell Netware

Transbase is delivered on several installation disks (3 1/2 "). The installation is started by execution of SETUP.EXE (Disk 1) from the file manager or from the program manager on any Netware client running MS Windows, Windows 95 or Windos NT. After a short initialization phase the dialogue oriented installation and configuration is performed.

The following parameters are requested by the installation:

- Installation-Directory Needed disk memory depends on the chosen volume of packages and varies from about 2 MByte until 15 Mbyte.

  The installation directory must be located on a netware filesystem which can be accessed from a NLM directly.

- Installation package(s) and corresponding license numbers.

The installation performs the following:

- Copying (with decompression) of the Transbase packages into the installation directory.
- Activation of the packages by patching licence numbers into specific files.
- Creation of file TBNLM.INI in the installation directory.

After installation, program SETUP.EXE can repeatedly be called from the Transbase installation directory to change or adapt package licence numbers.

12.3.1 Configuration

Some configuration parameters for Transbase are crucial for the effective and sound processing of Transbase.

12.3.1.1 Localisation of Parameters

During installation the TBNLM.INI file is located in the installation directory. To let prgrams find the TBNLM.INI file the parameter:

```
(CLIB_OPT)/P<path to tbnlm.ini>
```

must be added each time a Transbase NLM is loaded.

Note: The most important environment component of Transbase is the variable TRANSBASE.

12.3.1.2 Transbase Parameters

At installation time, the file TBNLM.INI is created with corresponding entries. Its parameters can be changed either by a reconfiguration (call of SETUP.EXE in the installation directory) or manually with a normal text editor e.g. notepad.exe). Note that section name as well as parameter name are case insensitive, however the parameter value I s case-sensitive!.

An example of file TBNLM.INI is shown below, together with a short explanation of parameters. A more detailed explanation is given in the corresponding chapters.

```
[TBNLM]
TRANSBASE=sys:\TB    // Transbase Installation Directory

// Overriding service names (of file SERVICES)
// refers to TCP/IP communication
TRANSBASE_SERVICENAMES=tbkernel:tbserver

// alternatively: direct definition of port numbers
;TRANSBASE_SERVICENAMES=2024:2025
```

Note: The parameter TRANSBASE is mandatory Unless the service names for network communications are directly defined via the `TRANSBASE_SERVICENAMES` variable, the file SERVICES are searched for entries of the form:

- tbkernel `<port>/<protocol>`
- tbserver `<port>/<protocol>`

12.3.1.3 Communication

The only available communication protocol is TCP/IP. This protocol accesses entries in the following files:

- File HOSTS

  This file holds a mapping between symbolic host machine names and Ethernet addresses.

  A Transbase client connects to a database by supporting a database name in the form:

```
<database>@<host>
```

The hostname can either be a symbolic host name (which then is resolved via the file HOSTS) or directly the ethernet address (e.g. 192.9.200.9).

- File SERVICES

  Via this file, all available network services are listed and their corresponding port addresses are defined.

  The following entries are relevant for Transbase:

```
...
tbkernel    \verb#<tbk_port>/tcp#
tbserver    \verb#<tbs_port>/tcp#
...
```

  These services must have been inserted both on client and server machine with consistent port addresses.

  The variable `TRANSBASE_SERVICENAMES` in file TBNLM.INI can override the above service names or directly define port addresses.

Chapter 13

Trouble Shooting

This chapter describes the most common error situations and discusses reasons possible remedies.

DLL TBxxxx.DLL was not found or cannot be loaded A list of requested runtime DLLs and the corresponding communication DLLs can be found in Appendix A. Since Transbase DLL's are not copied to the Windows directory, the OS might not found a Transbase DLL when invoking an application not from the Transbase but from another directory, which is normally the case when starting third party products. In that case one of the following must be true:

- The Transbase directory must be part of the PATH variable
- The current working directory must be the Transbase directroy. This can be arranged by defining an icon in the program manager.

Wrong configuration for remote database access The application has tried to CONNECT to a non-local database (`<database>@<host>`), but Transbase has been configured for Linked-In communication.

Configure a network protocol via the SETUP-program in the installation directory or by editing file TBWIN.INI (parameter HOSTACCESS16 bzw. HOSTACCESS32).

Unknown Host The application has tried to CONNECT to a non-local database (database name: `<database>@<host>`), and no host with the specified name could be found.

Specifiable hosts are those listed in file HOSTS of the network configuration or valid Ethernet addresses.

Server not reachable Verify that Transbase daemons tbserver, tbkernel or tbmux have been started. The network file SERVICES must contain these

services with corresponding port numbers, this holds for client and server machine.

An alternative is to define service name or port address directly via variable `TRANSBASE_SERVICENAMES`.

Another cause for trouble may be that a Transbase kernel (on server side) did not find its end and listens to its client though this client has abnormally ended by a crash. In this situation, the port address remains occupied by the clientless kernel.

To identify all kernels, issue the command

```
tbadmin -i <dbase> connections
```

which gives a detailed list of all connections to this database. Kernels without clients can be killed (UNIX-Kommando : `kill -9 <pid>`)

Database ... does not exist A Transbase kernel running on a host H identifies all databases existing on H via the file **F.DB** which itself is addressed via the configuration parameter TRANSBASE.

The following fields are relevant in this context (see a detailed description in Manual System Guide):

1. Database Name
   name of the database, locally unique, case sensitive.
2. Database Directory location of the database directory (holding file TBDESC, directories BFIM, DISKS etc.)
3. Host name
   Not evaluated
4. Database Type
   - 1 / 65 Transbase/Myriad Standard-Database
   - 2 / 66 Transbase/Myriad Editorial-Database
   - 3 / 67 Transbase/Myriad CD-ROM-Database
5. Database-Id
   Locally unique database identifier.
6. Bootflag
   Only evaluated on multi-user databases

Database ... not booted Multi-user databases must be booted before database processing (Command: tbadmin -b database)

Error in File TBDESC The file TBDESC in the database directory of a specific database contains various description parameters of the database, e.g.

directories, paths of database files (TBDSKxxx) and possibly paths of CD-ROM-Files (RFILExxx).

It may happen that some of these pathnames have become invalid by manual copying, renaming, configuration changes, etc.

Furthermore, it must be assured that the version of the database system (in detail: version of program tbadmin at database creation time) and the version of Transbase kernel match.

Unreasonable Error Message or Crash in Error Situation This occurrs is in most cases due to a incompatible file tberror.h which contains the textual error messages.Often this happens if more than one Transbase versions are installed on one machine.

File TBxxx.DLL already loaded With the exception of Windows NT 95, DLLs do not allow multiple instantiation.

Only one Transbase application at a time may run on other Windows platforms.

Server already started With the exception of Windows NT 95, Transbase server processes cannot be multiply instantiated. This means that only one server can be running here on one machine and also only one database can be accessed at a time.

Database ... not booted

Sometimes this error message is caused by DLLs which have not been unloaded.

Some DLLs are dynamically loaded at the first database access (CONNECT) and unloaded with the last database call (DISCONNECT).

If the DISCONNECT call is not executed (due to program crash, abort), these DLLs remain loaded and may cause an inconsistent program state at the next CONNECT- call. In general, the effects are not predictable and either of the following may be necessary:

- Restart of the Transbase application
- Restart of Windows (release of different ressources).
- In some cases the machine has to be rebootet (release of network resources, ports).

?

TBX already active By the tast switching mechanism, an application may get control and enter tbx although the last TBX call has not yet been finished.

With activated task switching, each application is responsible to avoid TBX calls if the TBX is already active (cmp. 6.7.1, 6.7.2).

Appendix A

Transbase Installation Files

A.1 Transbase Installation for Windows

Files	Description	Transbase Manual Keyword
Server		
tbker32.dll	linked in variant	
tbkwso32.dll	WinSocket-Server	System and Installation Guide: tbkernel
tbkdde32.dll	DDE-Server	System and Installation Guide: tbkernel
Runtime-DLLs		
tbpasc32.dll	Dynamically linked	
tbtasc32.dll	runtime libraries	
tbtype32.dll	type conversion routines	
tbtbx32.dll	communication, etc.	
Administration		
tbadm32.exe	Administration	System and Installation Guide: tbadmin
Communication		
tbbun32.dll	Linked_in kernel	
tbwsoc32.dll	WinSocket interface	
tbdde32.dll	DDE-interface	
Frontends		
wtbi32.exe	graphical TB/SQL-Frontend	
CD-ROM-Toolkit		
tbmkro32.exe		CD-ROM Database Guide: tbmkrom
Develop.-Toolkit		
tbx.h	Transbase-Include-File	Programming Interface TBX
tbcompat.h	Compatibility to older versions	Programming Interface TBX
sqlca.h	ESQL-Include-File	Embedded SQL-Programming Interface
tbx32.lib	Import Library	TB/X - Programming Interface TBX

Appendix B

Availability

B.1 Client

Client	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Communication					
16-Bit-Variant					
Linked_In	x	x	x	x	x
TCP/IP					
WinSocket	x	x	x	x	x
PC/NFS	x	x	x	x	
LWP	x	x	x	x	
PC/TCP	x	x	x	x	
DECNet					
PathWorks	x	x	x	x	
DDE					
DDE	x	x	x	x	x
NetworkDDE	x	x	x	x	x
32-Bit-Variant					
Linked_In			x	x	x
TCP/IP					
WinSocket			x	x	x
PC/NFS					
LWP					
PC/TCP					
DDE					
Local DDE				x	x
NetworkDDE				-	x

Note:

- The following holds: Transbase clients are available on all network systems which are based on the socket interface of Windows, i,e, which contain a file WINSOCK.DLL (this is a sufficient condition).

  For example, PC/NFS and PC/TCP support a WINSOCK.DLL, LanWorkPlace < 4.0 does not contain a WINSOCK.DLL!

- The Win32s extension automatically delivers WSOCK32.DLL as 32-Bit-Add-on for WINSOCK.DLL.

  This means that a 32-Bit-extension (Win32s) automatically adds 16-Bit-Transbase-Clients to existing 32-Bit-Transbase-Clients.

- Microsoft supports for Windows for Workgroups two additional TCP/IP network systems based on the WinSocket interface(see above):

  – MS TCP/IP for Windows for Workgroups
  – MS TCP/IP-32 for Windows for Workgroups (virtual device driver - 16Bit!)

B.2 Server

Server	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Communication					
16-Bit-Variant					
Linked_In	x	x	x	x	x
TCP/IP					
WinSocket	x	x	x	x	x
PC/NFS	x	x	x	x	
LWP					
PC/TCP	x	x	x	x	
DECNet					
PathWorks					
DDE					
Local DDE	x	x	x	x	x
NetworkDDE		x		x	x
32-Bit-Variant					
Linked_In			x	x	x
TCP/IP					
WinSocket			x	x	x
DDE					
Local DDE				x	x
NetworkDDE				-	x

Note:

- Windows for Workgroups with Win32s-extension (WfW32) does not support network DDE-communication!
- Under PathWorks (DEC) no Server are available!

Appendix C

Instantiation and Connections

C.1 Single-Server / Multi-Server

Server-Instantiations Machine	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Communication					
16-Bit-Variant					
Linked_In					
tbker16.dll	1	1	1	1	1
TCP/IP					
tbkwso16.exe	1	1	1	1	1
DDE					
tbkdde16.exe	1	1	1	1	1
32-Bit-Variant					
Linked_In					
tbker32.dll	1	1	1	1	≥ 1
TCP/IP					
tbker32.dll	1	1	≥ 1	≥ 1	≥ 1
tbkwso32.exe	1	1	≥ 1	≥ 1	≥ 1
DDE					
tbkdde32.exe	1	1	≥ 1	≥ 1	≥ 1

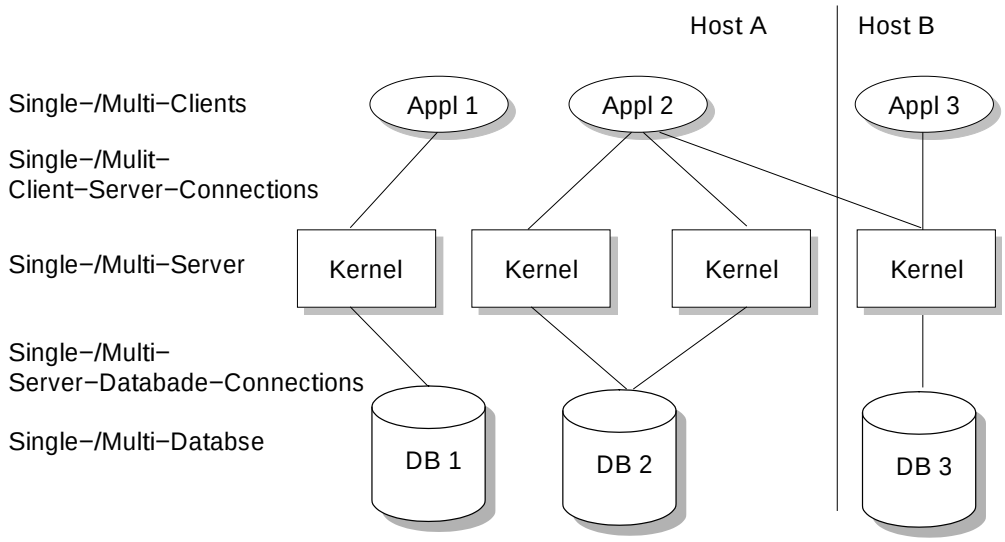

Table C.1: Single-Server / Multi-Server

C.2 Single-Client / Multi-Client

Client-Instantiations Machine	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Library					
16-Bit-Variant					
tbx.lib	1	1	1	1	1
32-Bit-Variant					
tbx32.lib	1	1	1	1	n

C.3 Single-Connection / Multi-Connection

Client-Server Connections	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Communication					
16-Bit-Variant					
Linked_In	1	1	1	1	1
Client-Server	n	n	n	n	n
32-Bit-Variant					
Linked_In	1	1	1	1	1
Client-Server	n	n	n	n	n

Note: A client simultaneously can have connections with the local server as well as with remote servers.

Of course, the number of simultaneous connections to local servers is induced by the instantiation restrictions of the servers on this platform.

C.4 Single-User-Database / Multi-User-Database

Server-Database Connections	*Platform*				
	16-Bit-System		32-Bit-System		
	Win16	WfW16	Win32	WfW32	WinNT
Communication					
16-Bit-Variant					
Linked_In					
tbker16.dll	1	1	1	1	1
TCP/IP					
tbkwso16.exe	1	1	1	1	1
DDE					
tbkdde16.exe	1	1	1	1	1
32-Bit-Variant					
Linked_In			x	x	x
tbker32.dll	1	1	1	1	≥ 1
TCP/IP					
tbker32.exe	1	1	1	1	≥ 1
tbkwso32.exe	1	1	1	1	≥ 1
DDE					
tbkdde32.exe	1	1	1	1	≥ 1